1991

Numerical Quadrature for General Two-Dimensional Domains

James V. Lambers

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
91-067

NUMERICAL QUADRATURE FOR GENERAL TWO-DIMENSIONAL DOMAINS

James V. Lambers
John R. Rice

CSD-TR-91-067
September 1991

Numerical Quadrature for General Two-Dimensional Domains

James V. Lambers*
John R. Rice
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
Technical Report CSD–TR–91–067
CAPO Report CER–91–32
September, 1991

Abstract

An algorithm is presented which computes integrals of functions over general two-dimensional domains with piecewise smooth boundaries. The algorithm first computes an appropriate polygonal approximation to the boundary with vertices on the boundary. The integration is completed by first using one-dimensional quadrature algorithms between the polygon and domain boundaries and then by using two-dimensional quadrature over a decomposition of the polygon into triangles. The algorithm is applied easily to multiply connected domains.

The algorithm is tested on a set of 52 domains obtained as instances from an extension of the set of domains of [9], [10] to 26 parametrized domains. Values believed to be accurate to 13 decimal digits are given for the areas of these 52 domains.

Categories and Subject Descriptors: G.1.4. [**Numerical Analysis**]: Quadrature and Numerical Differentiation - *adaptive quadrature*;
G.4 [**Mathematics of Computing**]: Mathematical Software;
G.m[**Mathematics of Computing**]: Miscellaneous - *FORTRAN*

General Terms: Algorithms

Additional Keywords and Phrases: Integration, two-dimensional domains, general shapes, multiply-connected

*Research supported in part by the National Science Foundation Research Experience for Undergraduates program, CCR 86–19817.

1 Introduction

This paper presents an algorithm for computing the integral of a function $f : D \subset R^2 \rightarrow R$ where D is a general domain. The algorithm assumes that ∂D, the boundary of D, is represented by explicitly parameterized pieces

$$C_i = \{(x, y) | x = x_i(p), y = y_i(p), p \in I_i = [a_i, b_i]\} \qquad i = 1, 2, \ldots, n$$

where the parameter functions $x_i(p)$ and $y_i(p)$ are continuously differentiable. In principle, all domains D with piecewise smooth boundaries can be represented this way but it is not always practical to obtain the functions $x_i(p)$ and $y_i(p)$.

Our objective is to obtain a robust quadrature algorithm by reducing the problem

$$\int\int_D f(x, y)$$

to a set of (a) one-dimensional quadratures of f with difficulties only due to the possible lack of smoothness in f, and (b) quadratures of f over triangles. We first describe the algorithm for a simply connected domain D. The approach is to approximate ∂D by a polygon P and then (a) integrate f over the area between ∂P and ∂D, and (b) integrate f over P. We show how to efficiently find an appropriate polygon P and how to use existing robust algorithms for stages (a) and (b). We then extend our algorithm to multiply connected domains in Section 4.

The results of testing the algorithm are presented in Section 5. It is shown to be capable of obtaining highly accurate results for a wide variety of simple and complex domains. Appendix A presents figures of the domains used, there are 26 parametrized domains and two cases were chosen for each to make a total of 52 test cases. Appendix B illustrates the operation of the algorithm by providing some snapshots of its progress for one example.

A companion paper presents an implementation of this algorithm using library routines for the two quadrature steps, e.g., routines from the IMSL library [3] [4] or Quadpack [2] [8] plus either user-supplied derivatives of $x_i(p)$ and $y_i(p)$ or a library numerical differentiation routine, e.g., DERIV [4] from the IMSL library.

2 Polygonal Approximation

Figure 1 illustrates a domain D with an appropriate approximating polygon P. We present the constraints P must satisfy for our quadrature method and present an algorithm to create P. Note that all the vertices of P are on ∂D.

Let B be a region bounded by an edge of P and the boundary of D as shown in Figure 2. Ideally, we would like to integrate f over B using an iterated integral of the form

$$\int_{s_1}^{s_2} \int_{0}^{t(s)} f(s,t)\, dt\, ds \tag{1}$$

for a region like the one shown in Figure 2a where s varies along the edge of P and t is perpendicular to it. This would allow us to use the composition of two one-dimensional quadrature algorithms. Unfortunately, this approach is impossible since we do not know the function $t(s)$. Therefore, we perform a change of variables to yield iterated integrals of the form

$$\int_{x_1}^{x_2} \int_{y_1(x)}^{y_2(x)} f(x,y)\, dy\, dx = \int_{a_i}^{b_i} \int_{\ell(p)}^{y(p)} f(x(p),y)\frac{dx_i(p)}{dp}\, dy\, dp \tag{2a}$$

or

$$\int_{y_1}^{y_2} \int_{x_1(y)}^{x_2(y)} f(x,y)\, dx\, dy = \int_{a_i}^{b_i} \int_{\ell(p)}^{x(p)} f(x,y(p))\frac{dy_i(p)}{dp}\, dx\, dp \tag{2b}$$

where $\ell(p)$ denotes the appropriate point on the straight line bounding the area (joining $(x(a_i), y(a_i))$ and $(x(b_i), y(b_i))$). All of these values can easily be computed with the information at hand.

However, if the vertices of P are not chosen carefully, difficulties can arise when attempting to integrate over B. Figure 3a shows an example where none of these integrals can be computed easily. The integral (1) cannot be computed accurately because neither dy/dx nor dx/dy is bounded in the domain of integration. Similarly with (2a) and (2b), neither $x_i'(p) = dx_i(p)/dp$ nor $y_i'(p) = dy_i(p)/dp$ is bounded in the domain of integration. Thus the polygon P must be such that either a horizontal or a vertical direction of outer integration can be used throughout without encountering a singularity.

It is easy to see that a region such as in Figure 3a can be avoided if we guarantee that either $x_i'(p)$ or $y_i'(p)$ does not have a zero on the interval $[p_a, p_b]$. One can employ a root-finding method such as regula-falsi for this purpose, but this can be expensive. There is an alternative which, although not foolproof, is very cheap and very effective.

We present an algorithm to partition a smooth piece C_i of ∂D into m arcs $A_1, \ldots, A_m$ such that for each $A_k, k = 1, \ldots, m$, the integral of f over the region defined by A_k and the line segment joining the endpoints of A_k can be computed accurately by an integral of the form (2b). We say that an arc $A_k \subset C_i$ is of type *horz* if it is horizontal (we use (2a)) and of type *vert* if it is vertical (we use (2b)).

We begin by partitioning I_i into M pieces of equal length, denoted $L_j, j = 1, 2, \ldots, M$, and construct the A_k by induction from the $Q_j = \{(x,y) | x = x_i(p), y = y_i(p), p \in L_j\}$ as follows. Let $L_j = [c_j, d_j]$. Assume that the original arcs $Q_j, j = 1, 2, \ldots, J$ have been grouped into arcs $A_k, k = 1, 2, \ldots, K$, and now consider the $(J+1)$st arc Q_{J+1}. It is either appended to A_K or becomes A_{K+1} depending on whether the constraints of the polygonal approximation to ∂D are met. We assume that A_K is of type *oldtype*, which must be *horz* or *vert*. Let $(x_1, y_1) = (x_i(c_{J+1}), y_i(c_{J+1}))$,

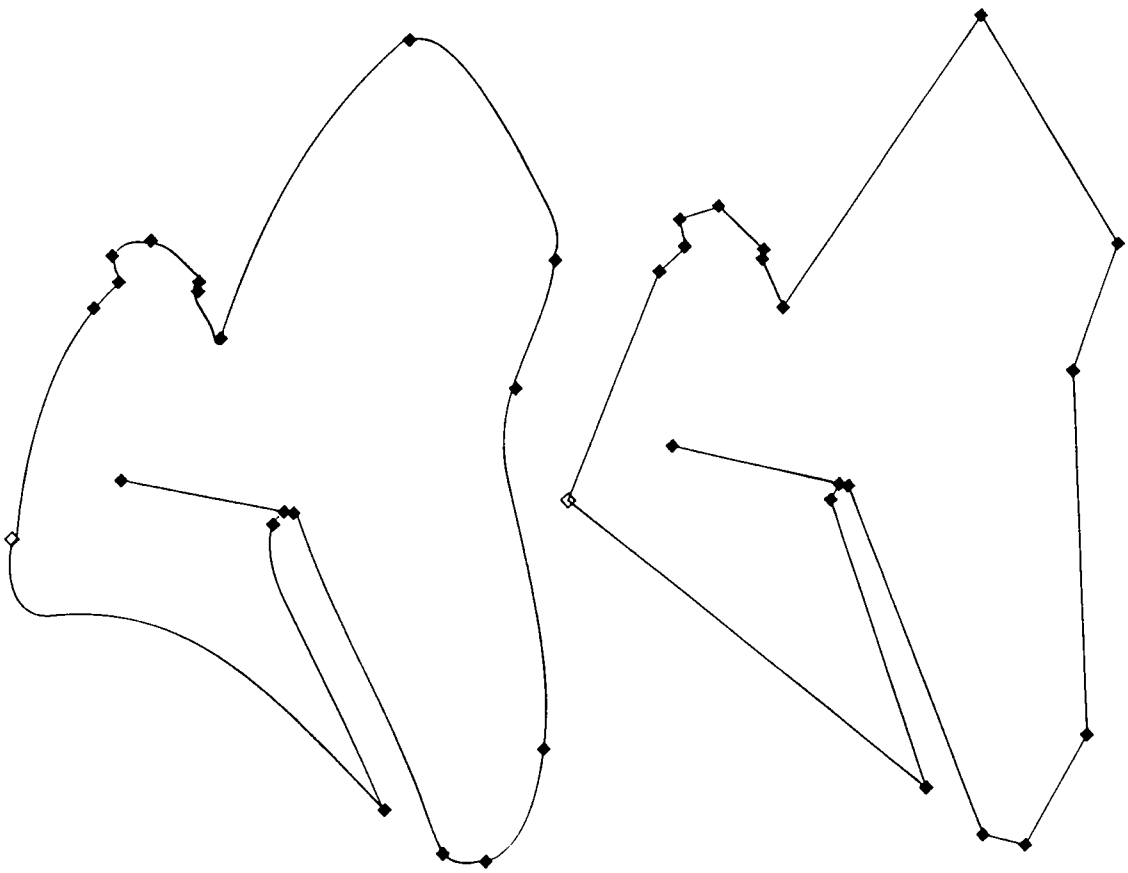


Figure 1: A domain D with an approximating polygon P.

$(x_2, y_2) = (x_i(d_{J+1}), y_i(d_{J+1}))$ be the coordinates of the endpoints of Q_{J+1} and let TOL be an algorithm parameter satisfying $1 < TOL < 2$. We then use

type $\leftarrow$ *oldtype*
if $|x_2 - x_1| > |y_2 - y_1| * TOL$ then
    type $\leftarrow$ *horz*
else if $|y_2 - y_1| > |x_2 - x_1| * TOL$ then
    type $\leftarrow$ *vert*
endif

if *type* $\neq$ *oldtype*
    begin a new arc $A_{K+1} = Q_{J+1}$ of type *type*
    $K \leftarrow K + 1$
else
    extend A_K to include Q_{J+1}
endif

This algorithm is designed to keep the y-slope bounded on any arc of type *horz*, and the x-slope of ∂D bounded on any arc of type *vert*. The type changes only when the slope goes outside the

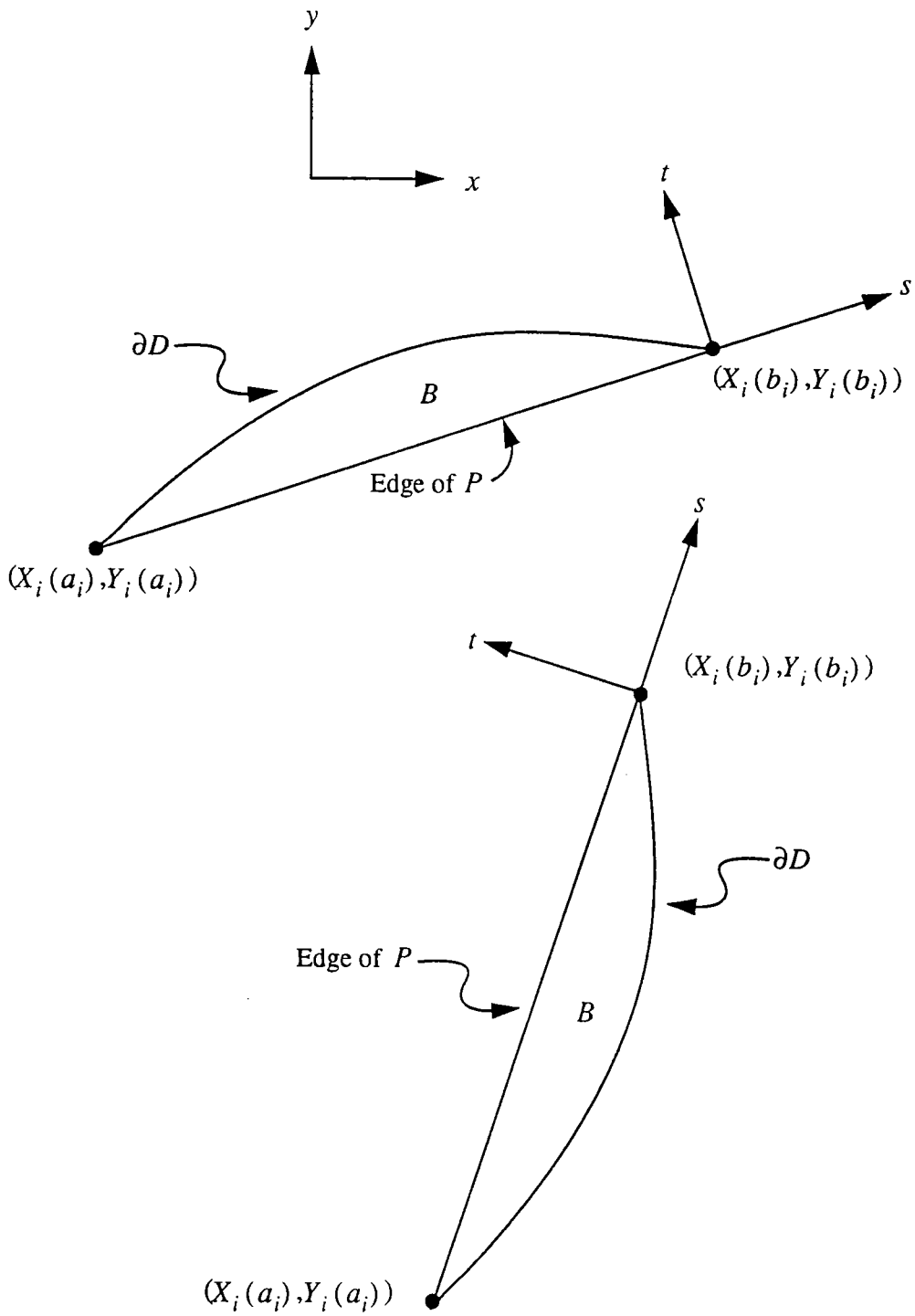


Figure 2: Areas B between an edge of P and ∂D with a generally (a) horizontal or x orientation or (b) vertical or y orientation.

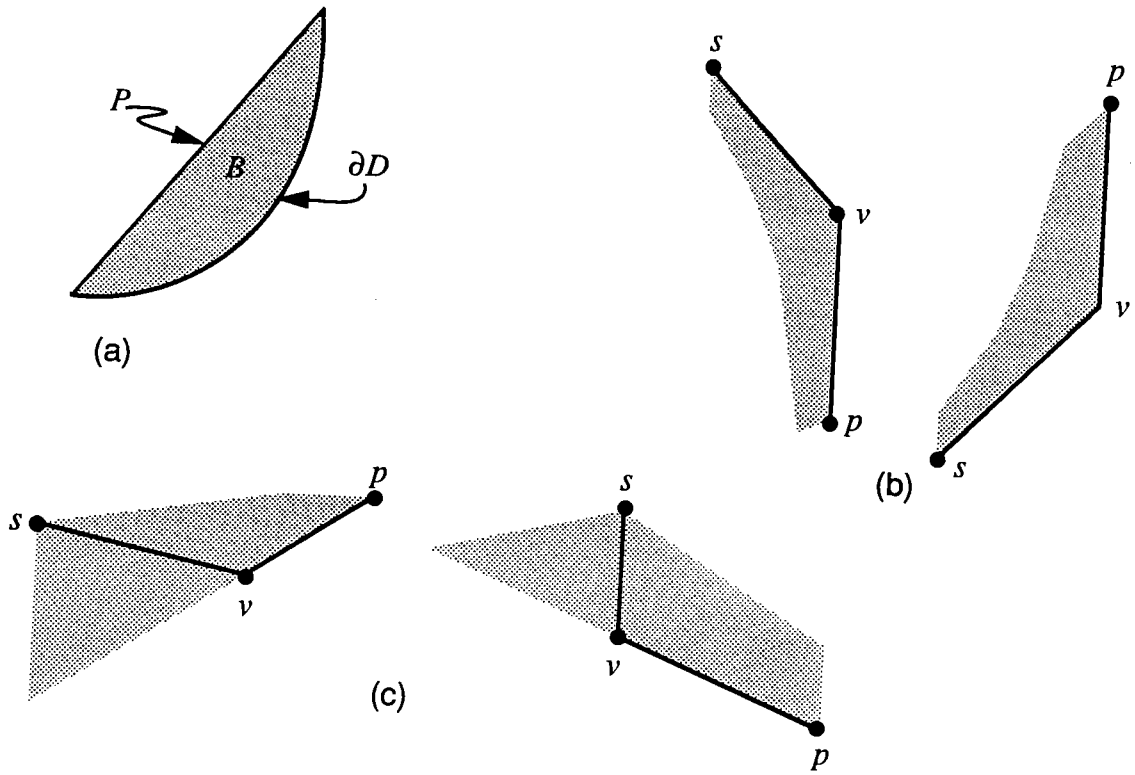


Figure 3: Situations along the boundary of a domain. (a) An area B between an edge of P and ∂D which has neither an x nor y orientation. (b) Situation for checking angle at v when $x_p = 0$. (c) Situation for checking angle at v when $x_p > 0$.

interval $(1/TOL, TOL)$. For the initial case, $J = 0, K = 0$, we set *oldtype* to *unknown* and use a similar procedure with $TOL = 1$ to establish the type of the arc A_1.

If the number M is sufficiently large, then each of the arcs A_k is of a form suitable for one of (1b) or (2b). The corresponding length $eps = d_j - c_j = (b_i - a_i)/M$ is similar to the *characteristic length* introduced by Rice in [11] to study adaptive quadrature algorithms. The role played by this length is to assure that over pieces of this length or less, the curve ∂D is simple, i.e., it does not have multiple inflection points or major changes in direction. One can formally prove that (a) there is an M for which this is so, and (b) if M is so chosen then the polygonal approximation P is such that (1b) and (2b) can always be used safely. We do not do this here.

In the implementation of this algorithm, the terminal point of each arc A_k, $k = 1, \ldots, m$, is a vertex of P. We store the following information about each vertex v of P:

VPAR(v)	The parameter value corresponding to v
IVPC(v)	The boundary piece on which v lies
VX(v), VY(v)	The x and y coordinates of v
IVTYPE(v)	The type, as defined above, of the arc between the predecessor of v and v.

This process is applied to each piece C_i of ∂D to create the polygon P. We note that the robustness of the integration is improved if every "symbolic corner" of ∂D is made into an end point of a piece. Sometimes one sees parameterizations where curves with different symbolic representations are joined up into one parameterized piece. For example:

$x(p) = p$
$y(p) = 0.0$
if $p > 3.141592654$ then
    $x(p) = p$
    $y(p) = 1. - \cos(p)$
end if

appears to define a continuously differentiable curve but round-off effects at the joining point $p = 3.141592654$ might cause unnecessary difficulty for the integration routines (or differentiation if numerical differentiation were used). Consider what might happen if 10 digits of accuracy is desired.

Once all vertices are chosen, we need to verify that the polygon is simple; i.e., that no two edges intersect except at a common vertex. All pairs of edges are inspected using a visibility algorithm which is described in the next section. If two edges intersect each other, the polygon is refined by using the following method for each of the offending edges:

$v_1 \leftarrow$ first vertex (in cyclic order) of the edge
$v_2 \leftarrow$ second vertex (in cyclic order) of the edge
$p_a \leftarrow VPAR(v_1)$
$p_b \leftarrow VPAR(v_2)$
$p_{new} \leftarrow 0.5 * (p_a + p_b)$
make new vertex v_{new} corresponding to p_{new}

3 Integration Over the Polygon

The remaining task is to integrate f over the simple polygon P. This is accomplished by partitioning P into triangles and integrating f over each of the triangles. The general algorithm follows:

while P has at least three vertices do
    find a vertex v of P such that
        a) the internal angle of v is convex (less than 180 degrees)
        b) the predecessor p of v (in the cyclic ordering of the vertices)
            is visible to the successor s of v.
    integrate f over the triangle formed by p, v, and s.
    delete v from P
endwhile

We now discuss in detail the algorithm for finding a vertex v that satisfies conditions (a) and (b). It follows from [7] that such a vertex v always exists. First, we outline a simple method for checking whether the internal angle of v is convex. Without loss of generality, we assume $v = (0, 0)$, since a simple translation of p and s reduces all other cases to this one. We also assume that P is oriented clockwise; the counterclockwise case is similar.

Let $s = (x_s, y_s)$ and $p = (x_p, y_p)$. There are three cases: $x_p = 0$, $x_p > 0$, and $x_p < 0$. The case $x_p = 0$ is illustrated in Figure 3b. From the figure, it is clearly seen that the internal angle of v is convex if and only if y_p and x_s are of opposite signs. Figure 3c illustrates the case where x_p is positive. In this case, the angle is convex if and only if s is in the shaded half-plane determined by the edge from v to p. The case where x_p is negative is similar.

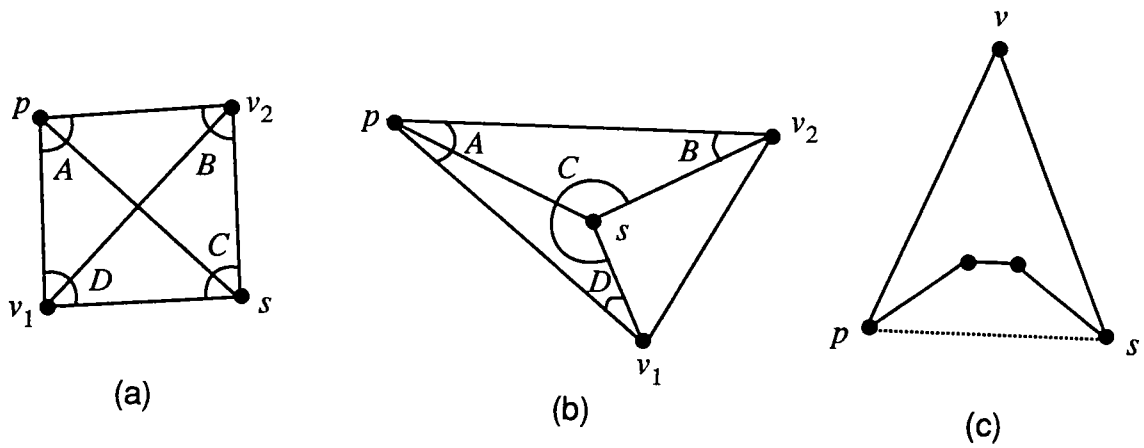


Figure 4: (a) The vertex p can "see" vertex s in the quadrilateral. (b) The vertex p cannot "see" vertex s in the quadrilateral. (c) The line segment joining p and s lies entirely outside P.

This method can be used to test whether v satisfies condition (b). To determine whether p can "see" s, we must first verify that no edge of P intersects the line segment between them. From Figure 4, we can see that such an intersection will occur if and only if the edge and the segment are the diagonals of a quarilateral. Let v_1 and v_2 be the vertices of any edge of P such that $\{v_1, v_2\} \cap \{p, s\} = \phi$. We check for an intersection by testing the convexity of the angles A, B, C, D seen in Figure 4. If one of the angles is not convex, then the polygon formed by the four vertices

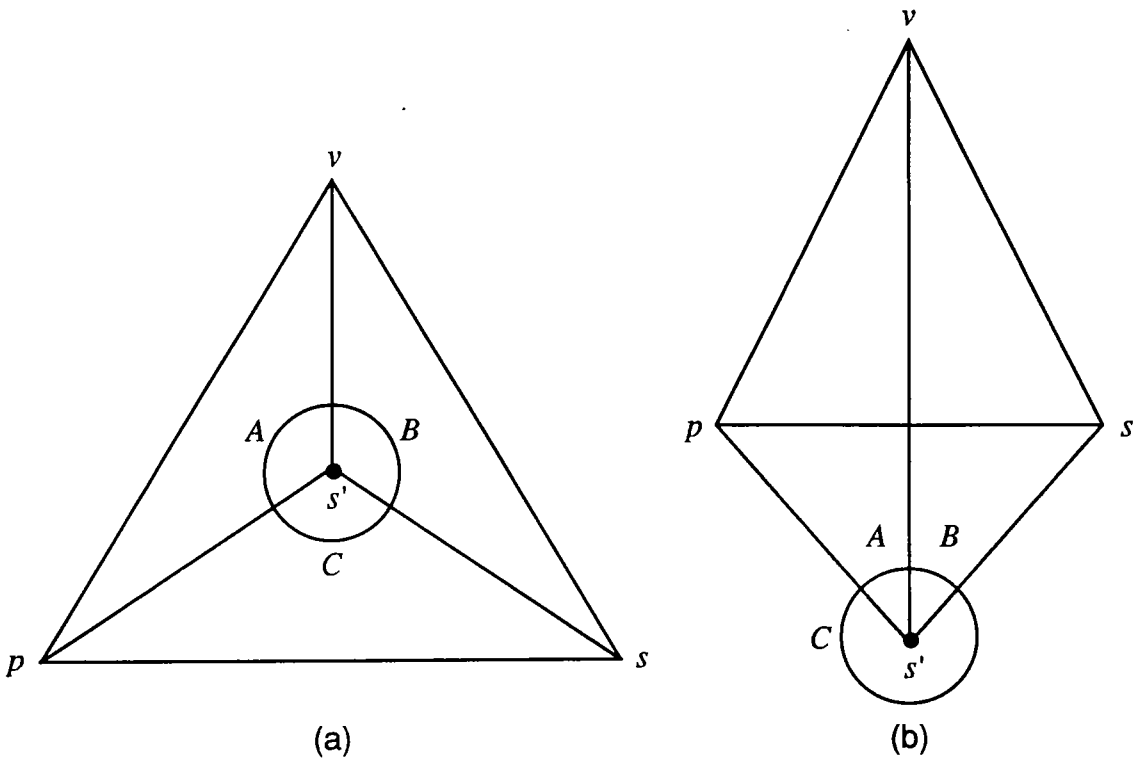


Figure 5: (a) The vertex s' lies in the triangle formed by p, v, and s. The angles A, B, and C are all convex. (b) The vertex s' lies outside the triangle formed by p, v, and s. One of the angles A, B, or C is obtuse.

is not convex and thus does not have diagonals.

Even if no edge of P intersects the line segment from p to s, we still do not know that p can see s. The line segment between them might lie entirely outside P. This situation is illustrated in Figure 4c. As the figure indicates, all vertices of P, except v, p, and s, must lie in a triangle formed by v, p, and s. Therefore we choose s', the successor of s, and check whether it is in this triangle. If s' is p, then P is a triangle and we are done. Otherwise, we must test the convexity of the angles A, B, and C shown in Figure 5ab. From this figure it is clear that s' lies in the triangle if and only if all three angles are convex.

4 Multiply Connected Domains

Our algorithm can be extended easily to domains that are not simply connected. Let $B_1, \ldots, B_n$ be the disjoint curves that together form the boundary of D, where B_1 is the boundary curve separating D from the unbounded region exterior to D. We then integrate f over D using the following algorithm:

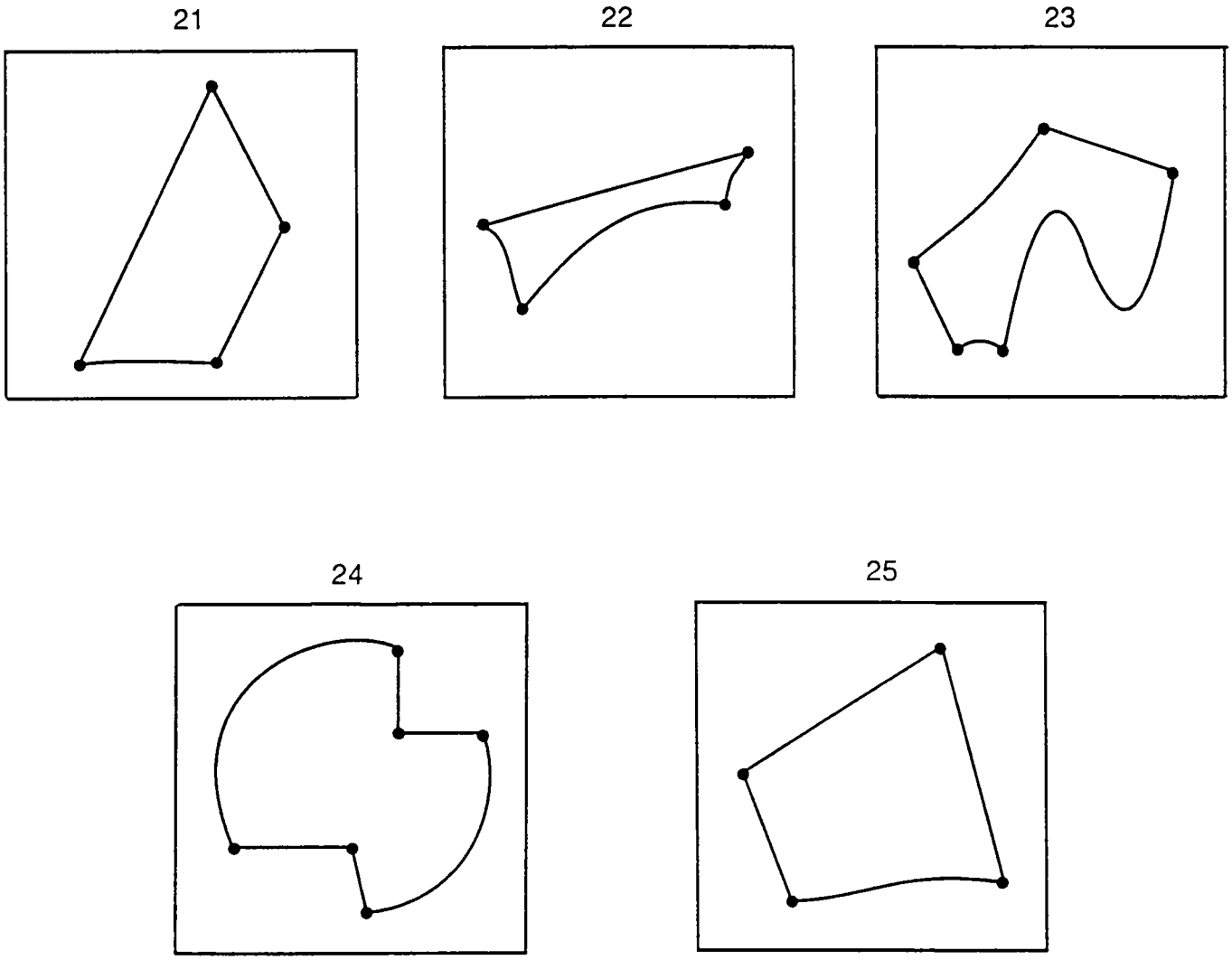


Figure 6: Instances of the five of the additional domains used to test the quadrature algorithm. The sixth domain is a small circle with a very long, very thin spike on it. The spike is so long compared to the size of the circle that the circle is barely visible (see Figure 7).

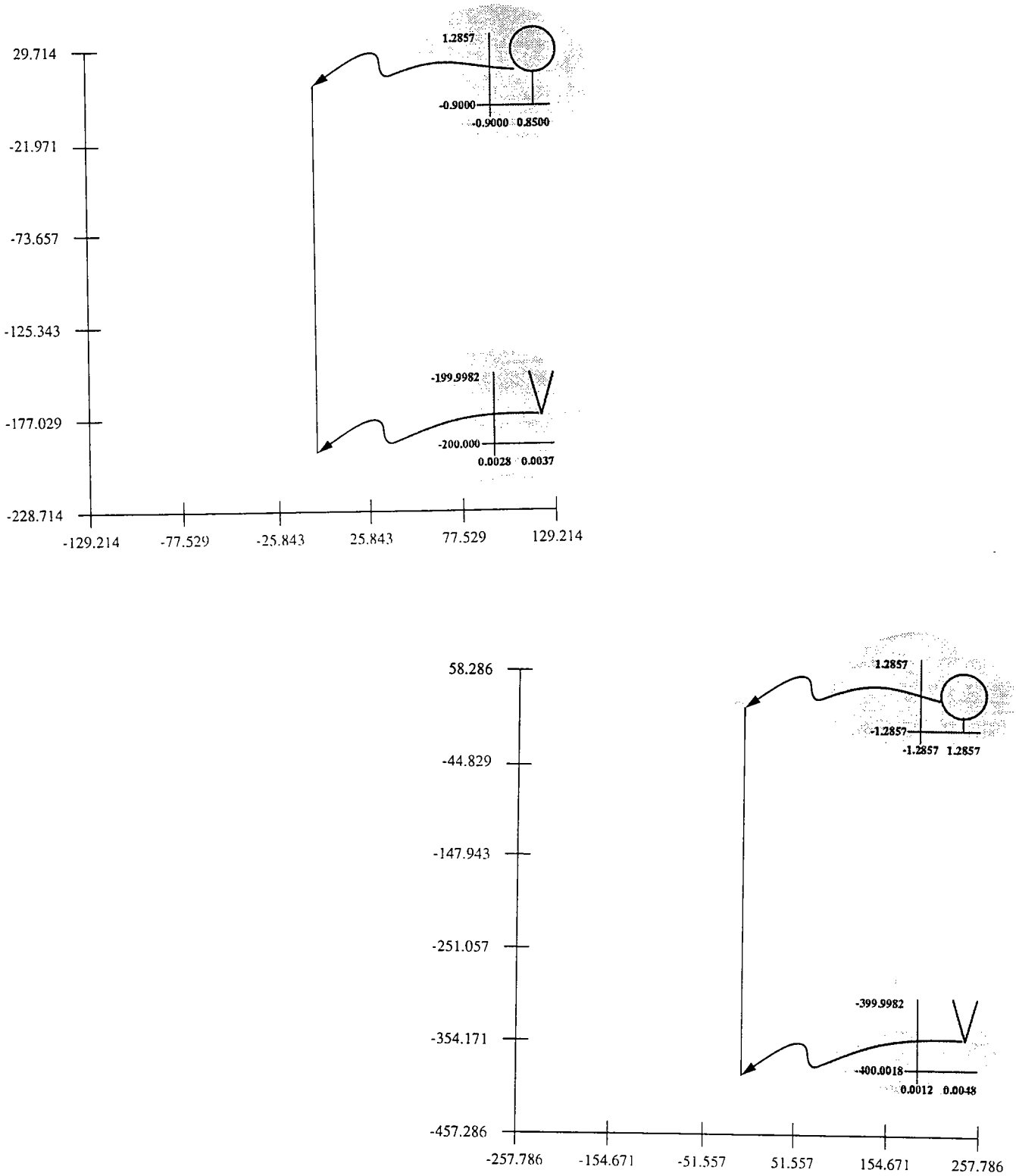


Figure 7: Two instances of domain 26, the two "ends" of the domains are magnified so the shape can be seen.

let D_1 be the domain with boundary B_1
let $TOTAL$ be the integral of f over D_1, using the algorithm for simply
  connected domains
for $i = 2, \ldots, n$ do
    let D_i be the domain with boundary B_i
    integrate f over D_i using the algorithm for simply connected domains
    subtract the value of the integral over D_i from $TOTAL$
endfor

This algorithm simply integrates f over D and its "holes", and then removes the contribution of the holes to the value of the integral.

5 Testing

One can test this algorithm for a variety of simple domains and integrands where exact answers are known. We have done this and find the algorithm capable of computing answers with accuracy within the expected round-off effects. This approach only tests simple domains so we have applied the method to compute the areas for the set of domains based on the set in [9], [10]. This set of domains has been enlarged from 20 to 26 domains since it was originally published; the six added domains are illustrated in Figures 6 and 7. Note that all of these domains are from a two parameter set of two-dimensional domains. Table 1 provides the areas of two instances of each domain in this set. We expect these values to be useful for those who improve upon this method in the future. Previous work on evaluating quadrature algorithms is found in [1], [5], [6], [12].

The accuracy of the integrations is checked by computing each area twice, one as given in the domain set and again with the domain rotated 21.6 degrees. The computations of the method are heavily influenced by the orientation of the domain and we believe the probability of accidental agreement of two incorrect results to be negligible. Thus we believe the values in Table 1 to be correct to 13 digits (14 digits are given). In no instance did we observe any unusual round-off effects.

Table 1: The areas of two instances for each of the 26 domains used to test the quadrature algorithm.

Domain	Parameters pairs and areas			Domain	Parameters pairs and areas		
Number	(a_1)	(b_1)	area	Number	(a_1)	(b_1)	area
1	0.8	0.6	0.48000000000000	8	1.0	1.0	3.1415926535461
1	0.22	0.88	0.19360000000000	8	-0.1	0.3	0.28274333881915
2	-2.0	0.5	0.25000000000000	9	0.2	0.2	23.960785565662
2	1.0	1.0	0.50000000000000	9	1.2	-0.3	23.324165793294
3	2.0	1.0	6.2831853071786	10	1.0	0.0	5.7853981633974
3	0.4	0.6	0.75398223686146	10	2.0	0.0	5.1415926535897
4	1.0	1.0	0.91000000000018	11	1.0	0.0	15.417187500002
4	0.5	1.0	0.91000000000018	11	1.0	0.2	11.030563245827
5	0.2	0.3	1.1872500000000	12	4.4	3.6	25.392126935530
5	1.4	0.5	0.77525000000000	12	8.0	4.0	37.231308926445
6	0.0	0.0	32.467499999978	13	0.25	1.0	3.0000000000000
6	-0.5	1.0	52.295843182848	13	0.1	3.0	3.6000000000000
7	0.4	0.2	0.25716814692889	14	0.0	0.0	0.35234517542809
7	0.6	0.1	0.30429203673222	14	1.0	0.0	0.57042850876159

Domain	Parameters pairs and areas			Domain	Parameters pairs and areas		
Number	(a_1)	(b_1)	area	Number	(a_1)	(b_1)	area
15	0.5	1.5	1.5707963267872	21	0.0	0.5	1.5106572878068
15	1.3	1.45	0.32397674239986	21	1.0	0.0	1.0106572878068
16	1.25	0.75	3.7795362321101	22	1.0	0.1	1.5402249253326
16	0.5	0.48	4.3011976922330	22	1.0	1.2	1.1026991039913
17	0.0	0.0	419.975673653780	23	0.0	0.5	3.5952039510777
17	1.0	0.5	944.945265721005	23	0.0	1.2	5.5230528666523
18	0.0	0.0	10.3345547673198	24	0.5	0.5	2.6189141552312
18	1.0	0.0	10.3345547673201	24	0.3333	0.1111	2.5641669941884
19	0.0	0.0	10.4550957239758	25	0.2	0.0	2.5727815531860
19	1.0	0.0	12.4405745260173	25	0.2	1.0	3.5727815531860
20	0.5	1.0	5.0548156085702	26	200.0	–	3.8084436789026
20	0.75	1.6	10.830020085346	26	400.0	–	4.4786457019282

References

[1] J. Casaletto, M. Picket and J.R. Rice, A comparison of some numerical integration programs, *Signum Newsletter* **4** (1969), 30–40.

[2] P. Favati, G. Lotti and F. Romani, Algorithm 691: Improving QUADPACK automatic integration routines, *ACM Trans. Math. Software*, **17** (1991), pp. 218–232.

[3] IMSL, *Math Library*, Sections 4.1, 4.2, IMSL, Houston, TX (1987), 561–604.

[4] IMSL, *Math Library*, Sections 4.4, IMSL, Houston, TX (1987), 625–627.

[5] D.K. Kahaner, Comparison of numerical qudrature formulas, in *Mathematical Software*, J.R. Rice, Ed., Academic Press, New York, 1971, pp. 229-259.

[6] J.N. Lyness and J.J. Kaganove, A technique for comparing automatic quadrature routines, *Comp. J.* **20** (1975), 170–177.

[7] Joseph O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, 1987.

[8] R. Piessens, et. al., *QUADPACK: A Subroutine Package for Automatic Integration.* Springer-Verlag, Berlin, 1983.

[9] J.R. Rice, Numerical computation with general two dimensional domains, *ACM Trans. Math. Software*, **10** (1984), 443–452.

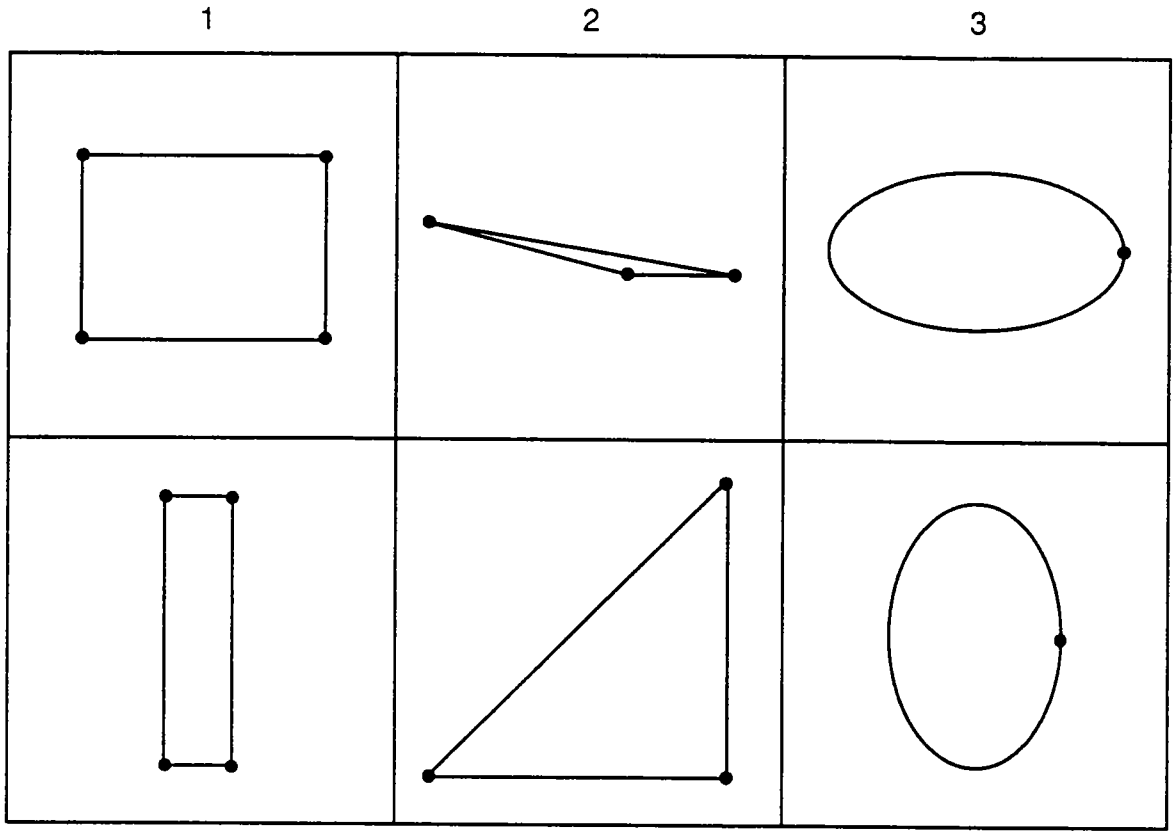
[10] J.R. Rice, Algorithm 625: A two dimensional domain processor, *ACM Trans. Math. Software*, **10** (1984), 453–462.

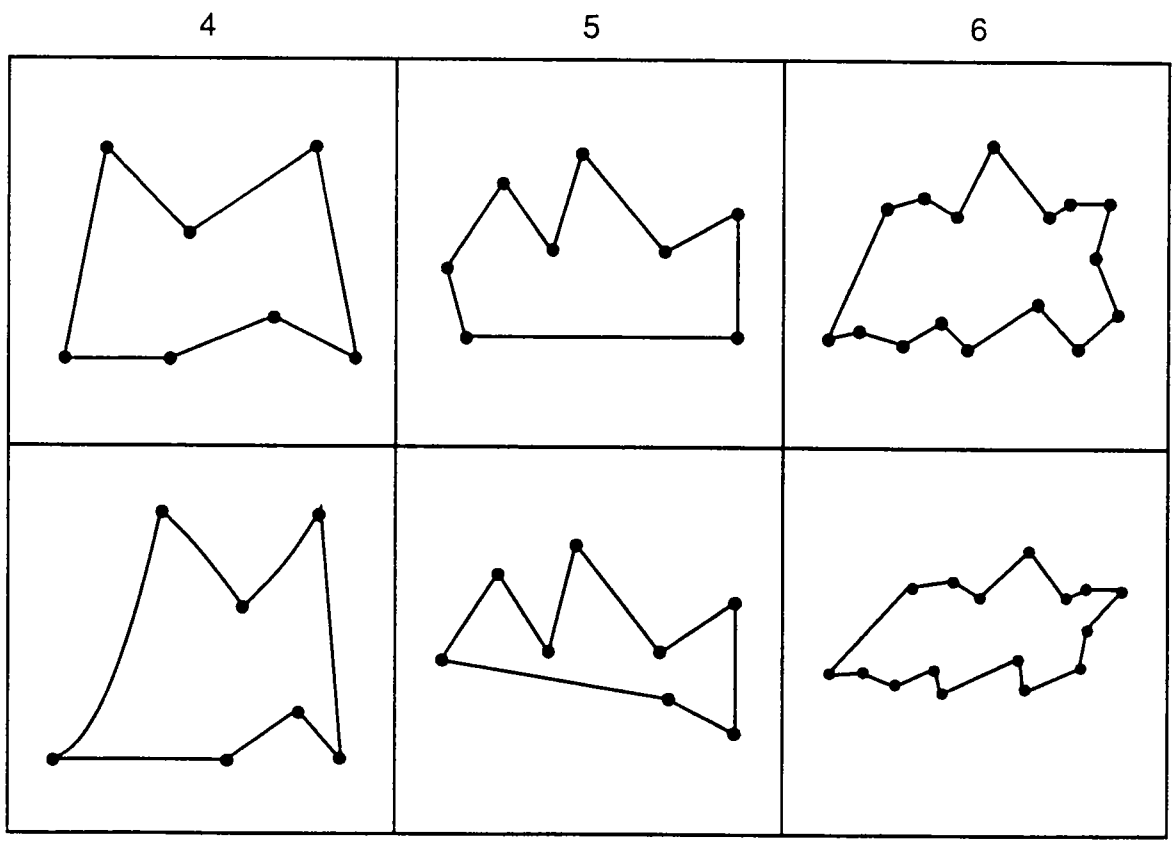
[11] J.R. Rice, A metalgorithm for adaptive quadrature, *J. Assoc. Comp. Mach.*, **22** (1975), 61–82.

[12] I. Robinson, A comparison of numerical integration programs, *J. Comput. Appl. Math.*, **5** (1971), 207–223.

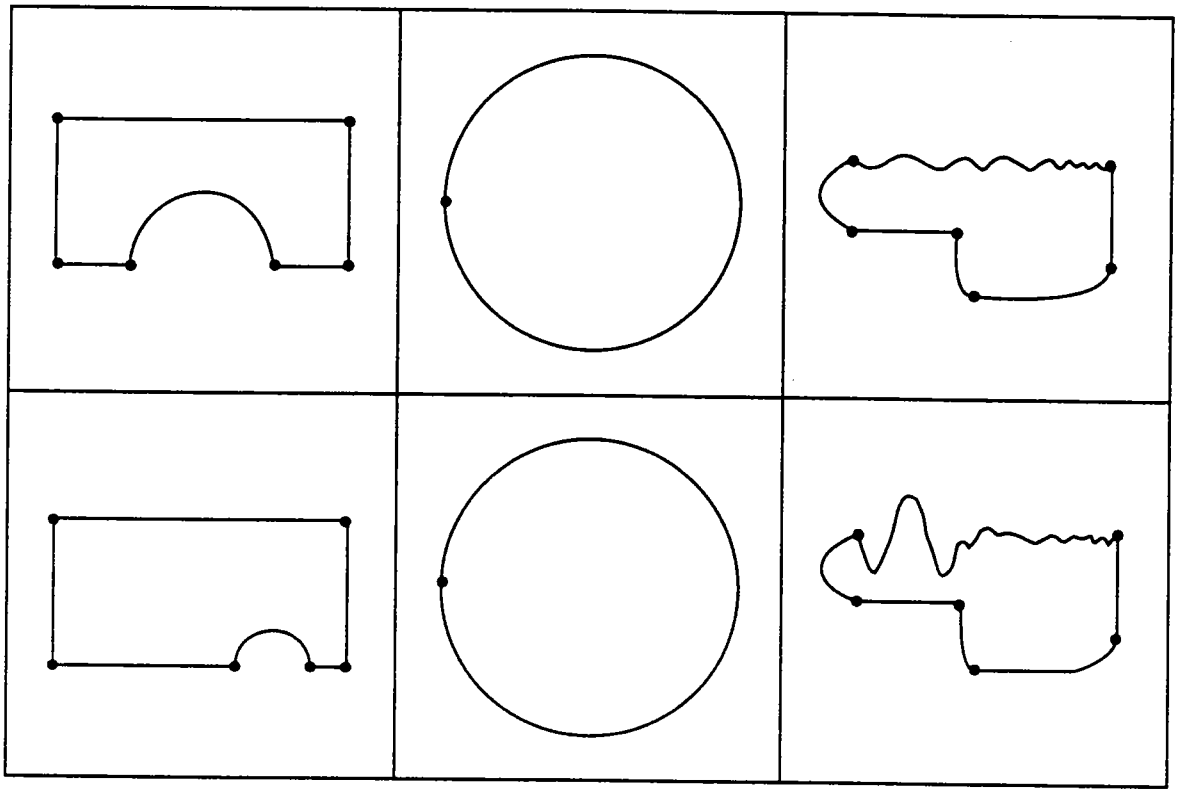
APPENDIX A

The 52 domains whose computed areas are given in Table 1 are plotted.

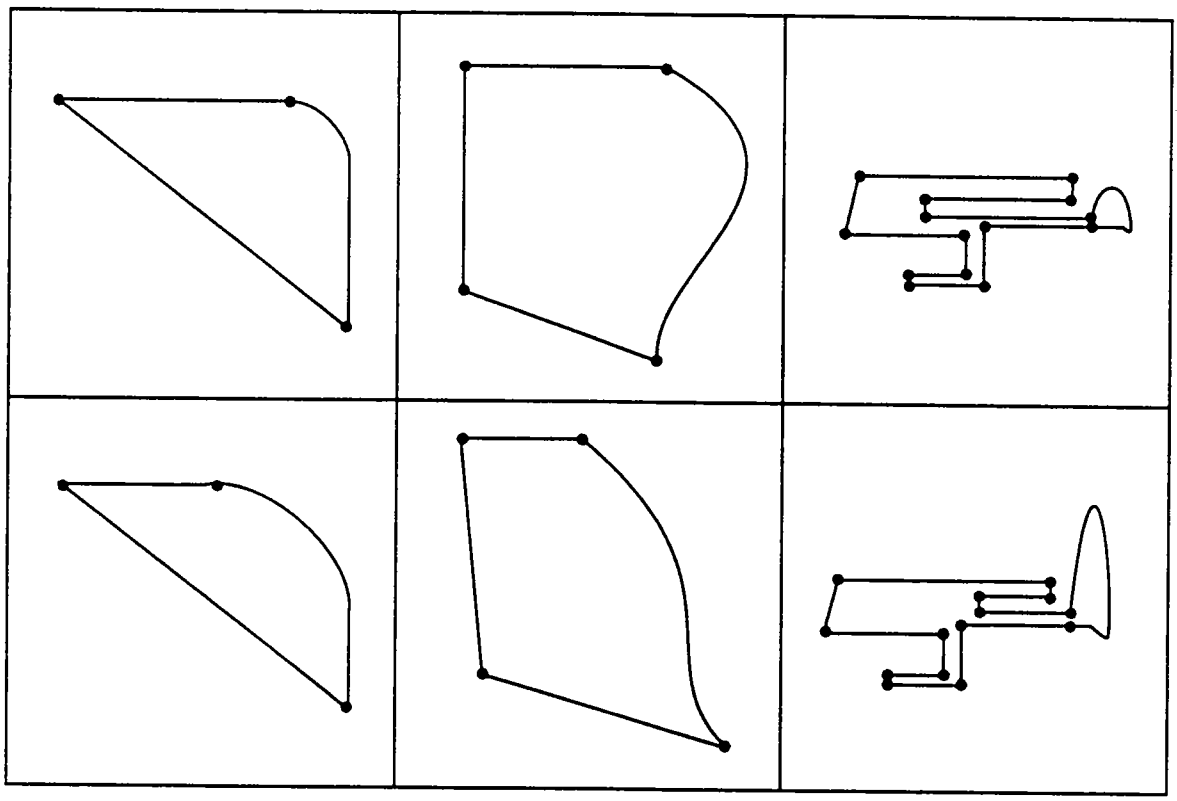




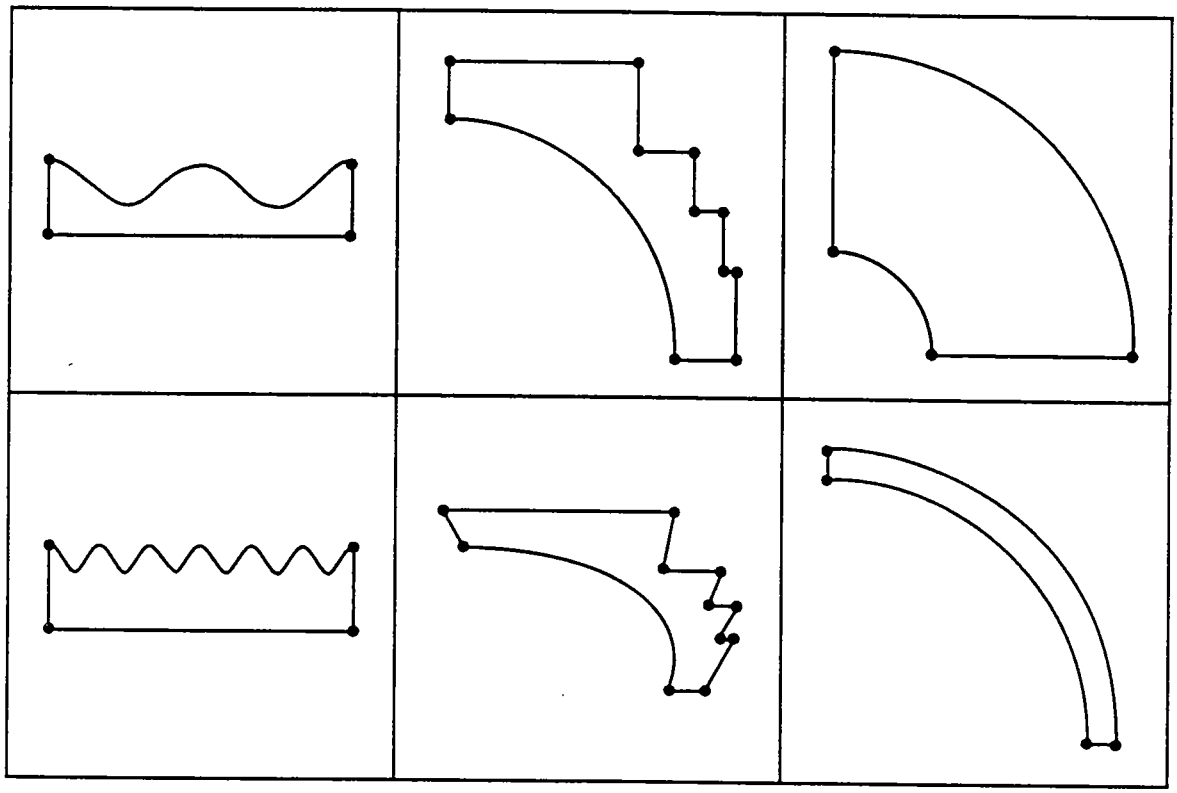
7 8 9



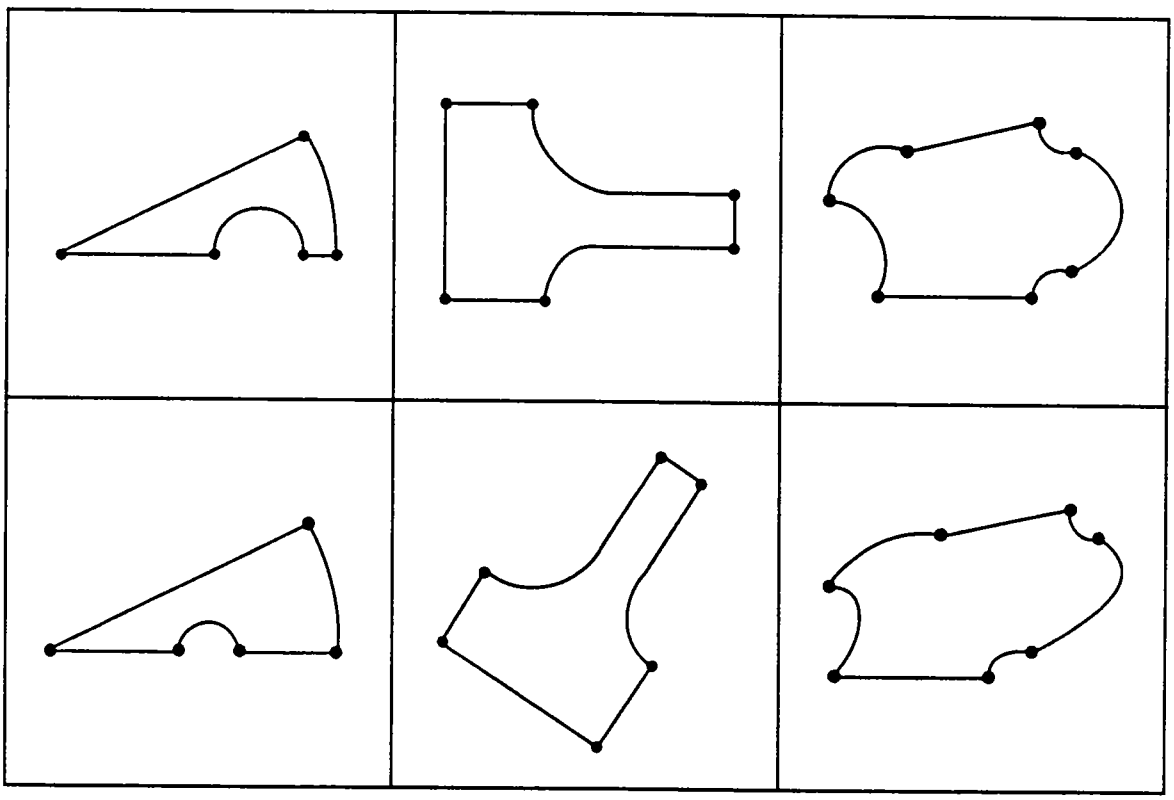
10 11 12



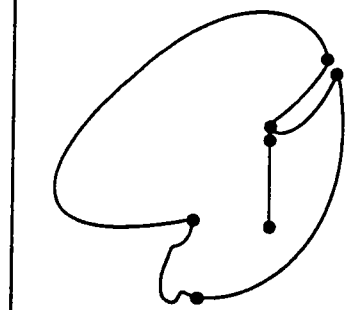
13 14 15



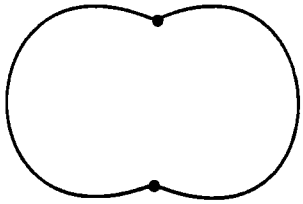
16 17 18



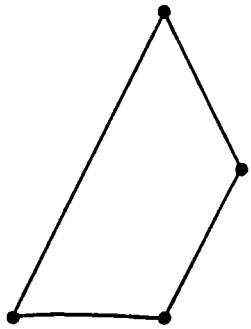
19

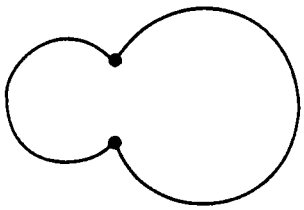


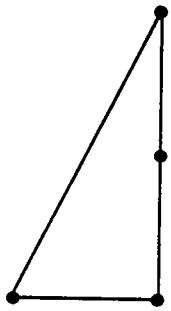
20



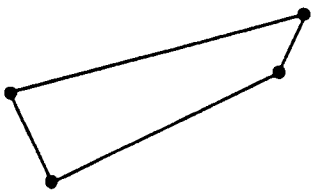
21



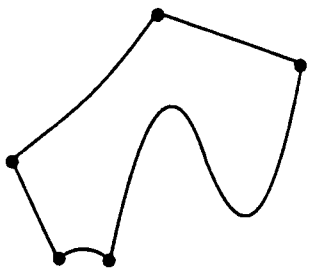




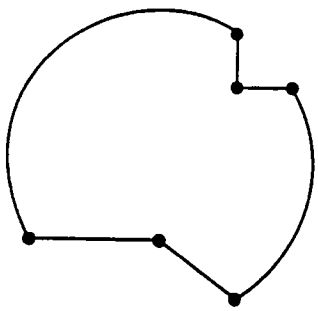
22

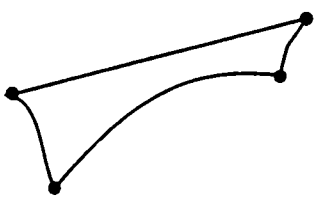


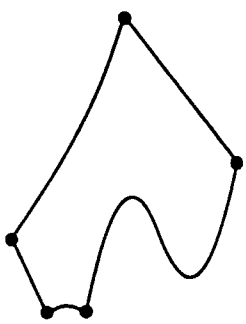
23

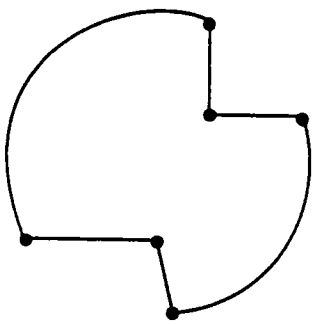


24

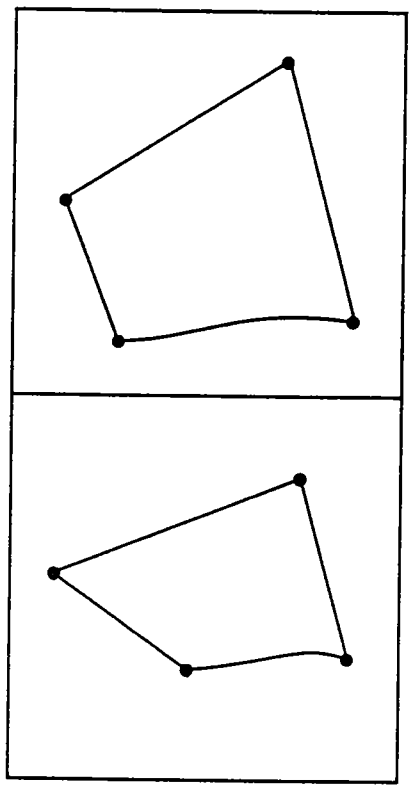


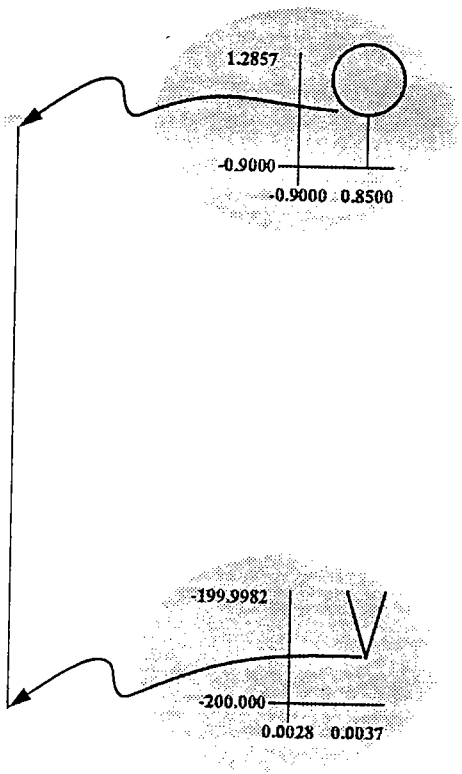






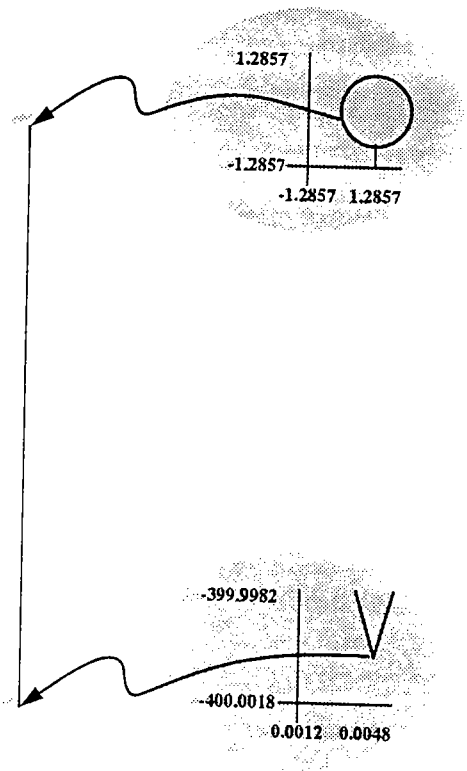
25





Domain 26

a = 200

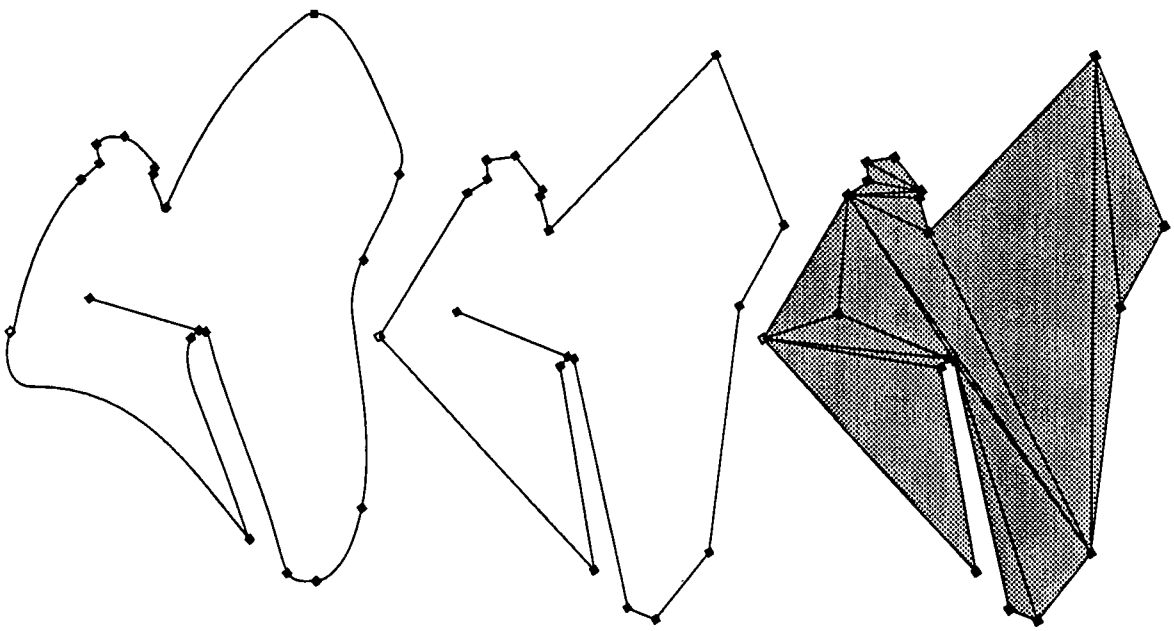


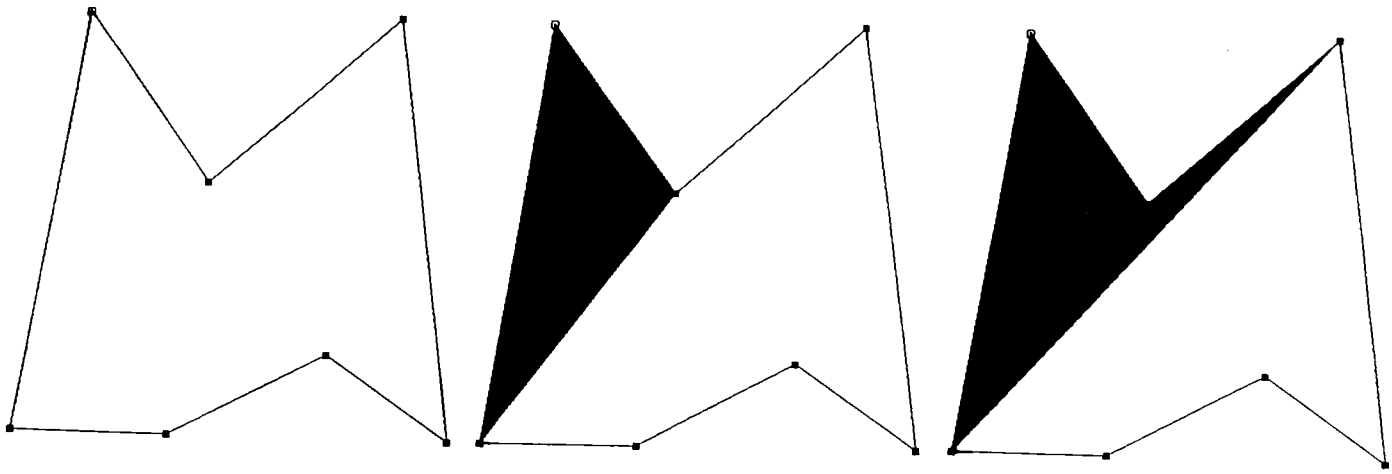
Domain 26

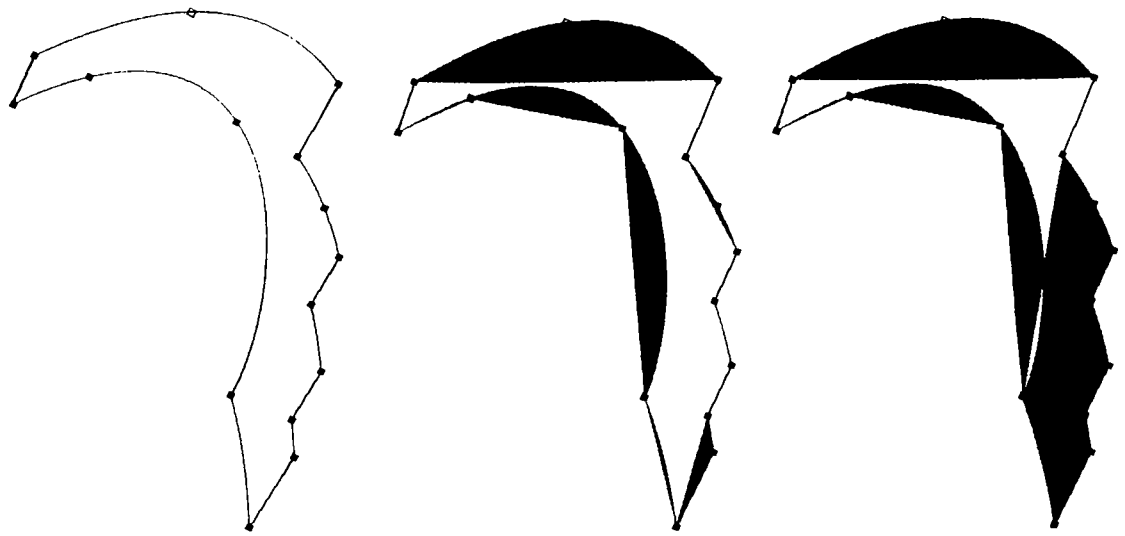
a = 400

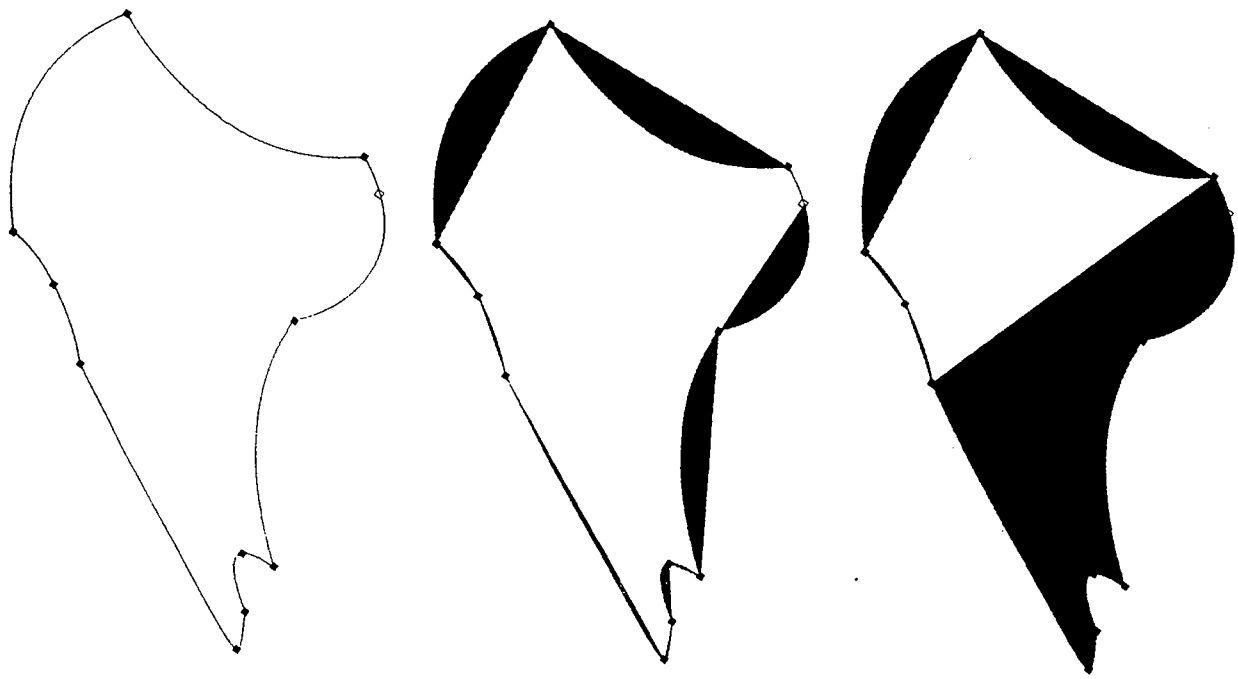
APPENDIX B

Snapshots of the algorithm are shown for an example itegrations. These snapshots are X-window dumps from a software tool for using the algorithm. This tool presents a new display as each subarea is defined. The first set of 3 snapshots shows the original domain, the approximating polygon, and all the triangles used to fill the interior. This set is an extension of Figure 1. The next three sets of 3 snapshots show the original domains, the triangles along the boundary, and part of the fill in the interior. The dots on the boundary in the first snapshop are the end points of the polygonal approximation to the boundary.



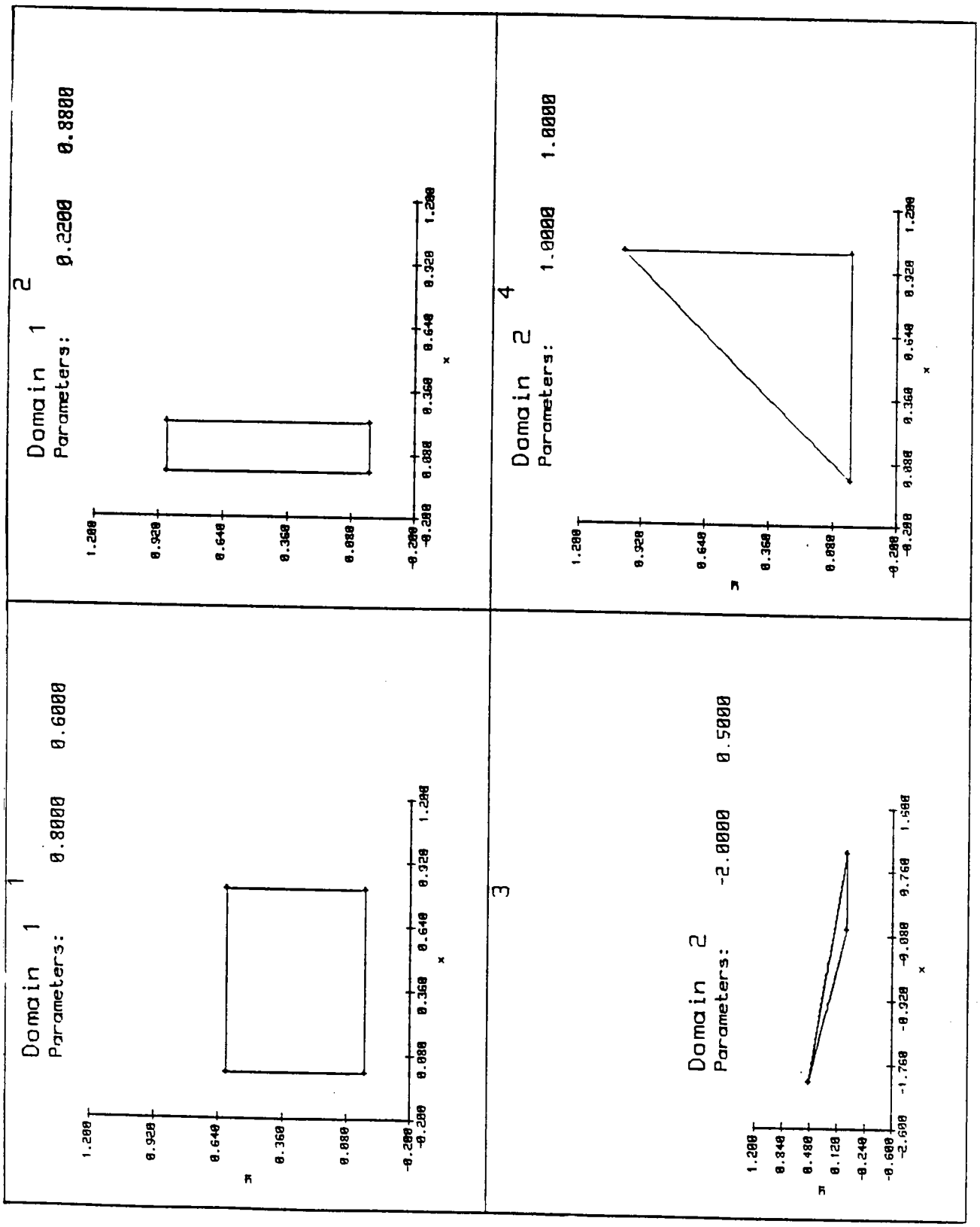


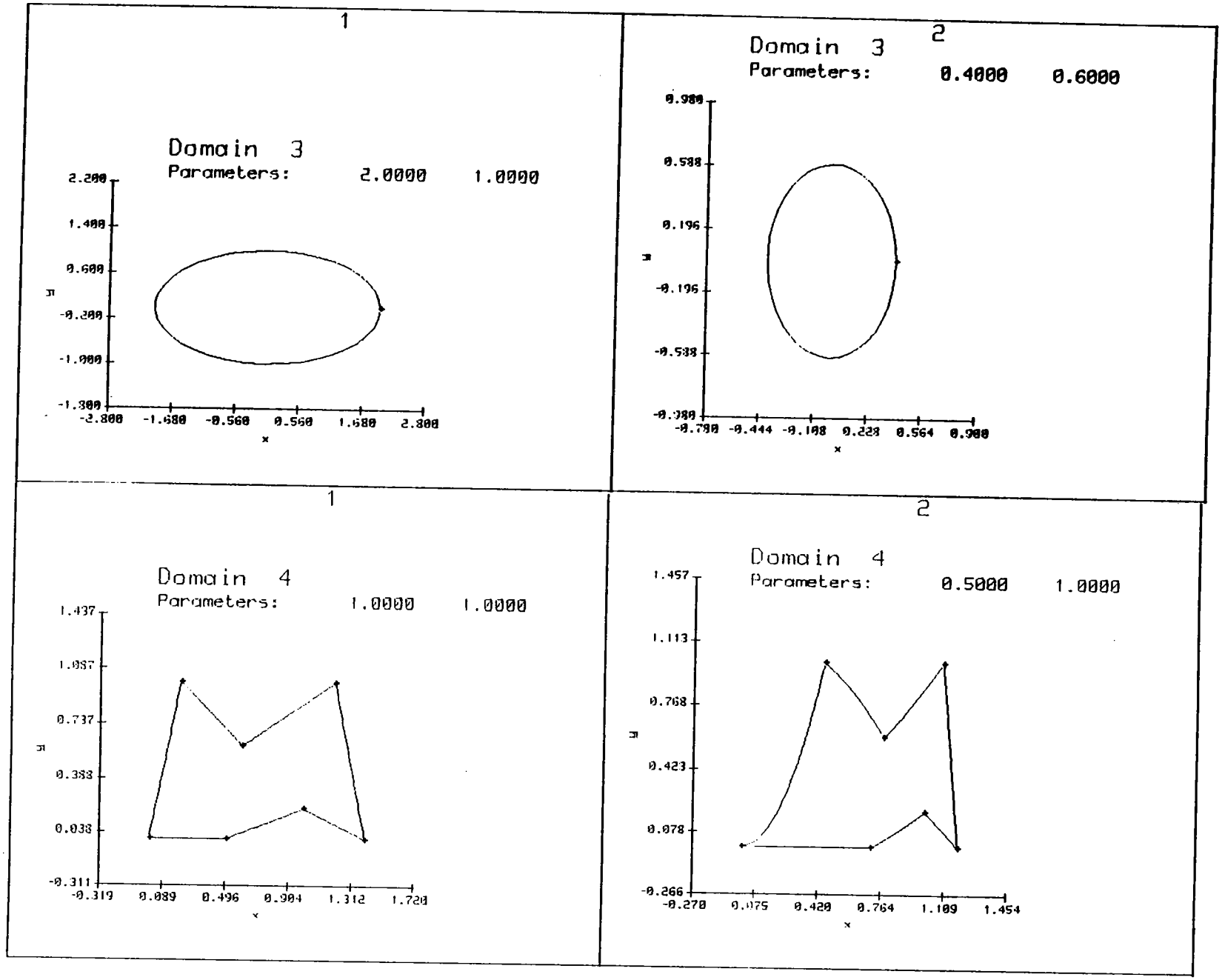


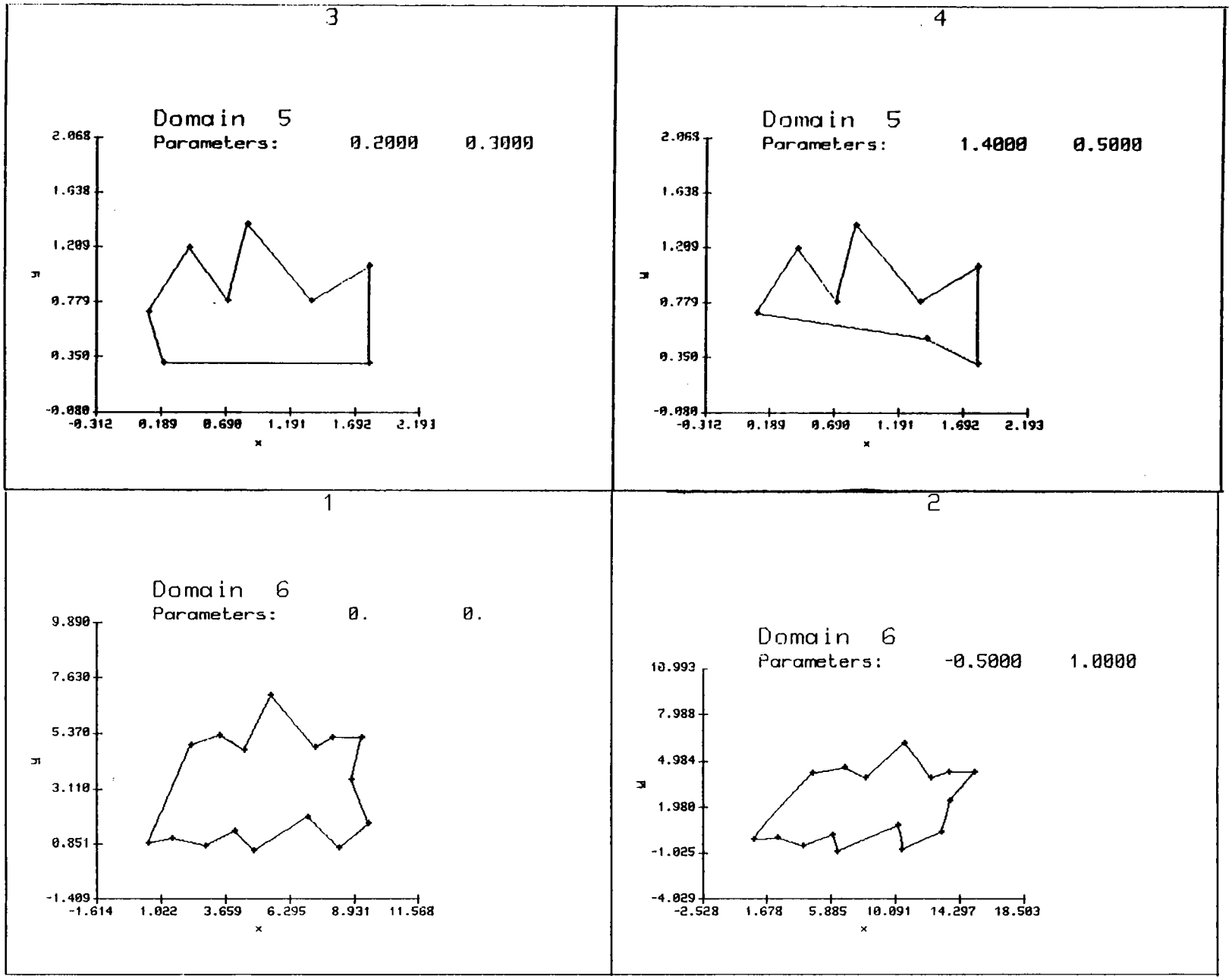


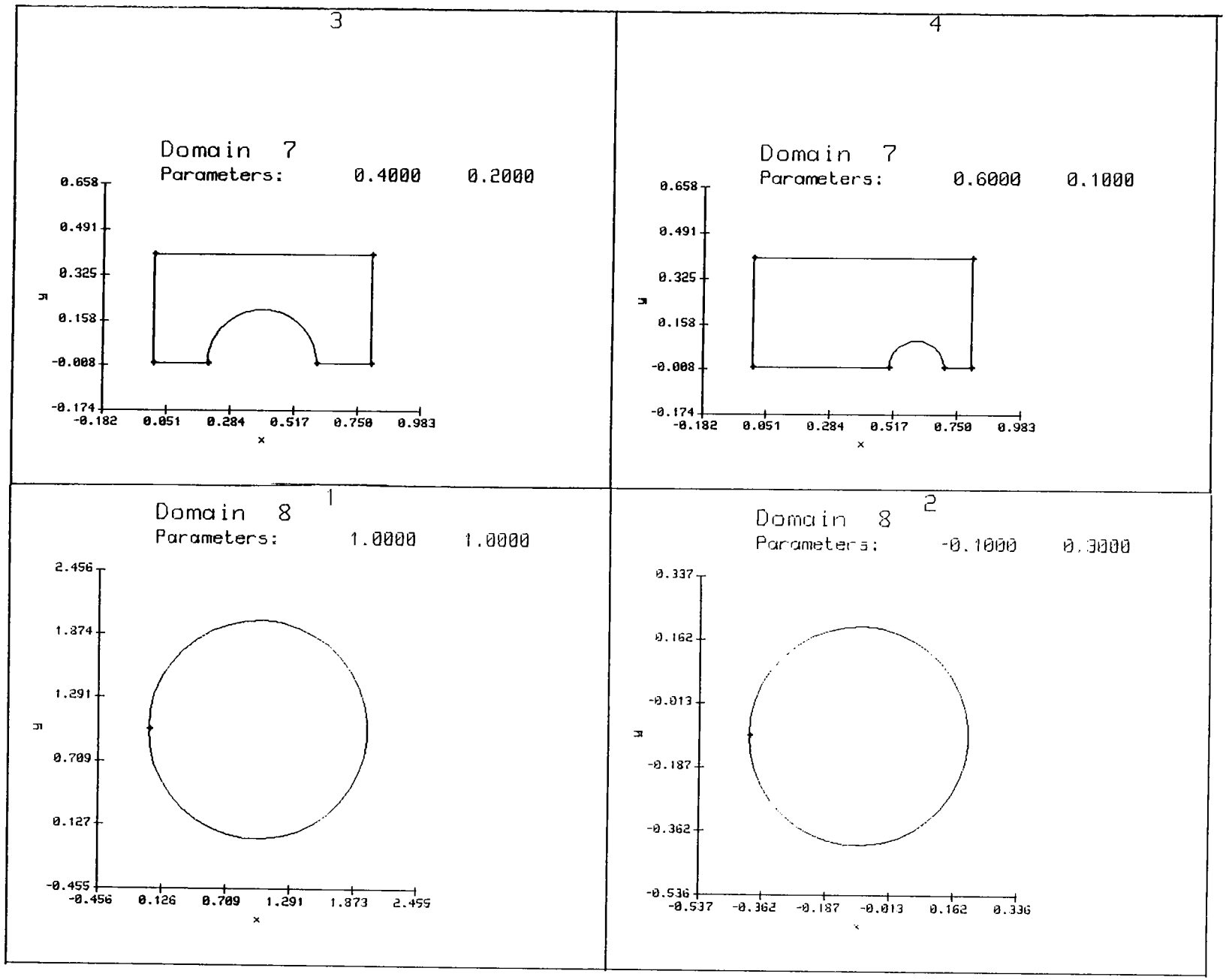
APPENDIX C

Plots of the 25 pairs of test domains are given along with coordinate axes.



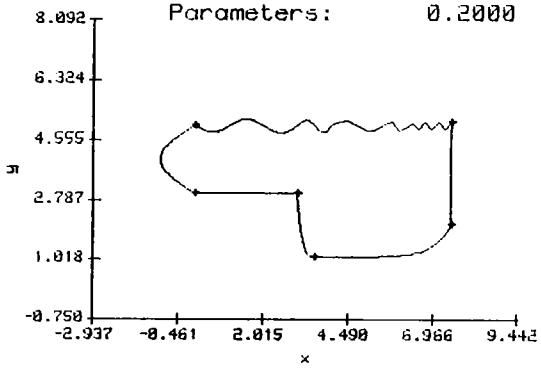

1
Domain 3
Parameters: 2.0000 1.0000
2.200
1.400
0.600
-0.200
-1.000
-1.800
-2.800 -1.680 -0.560 0.560 1.680 2.800
x
2
Domain 3
Parameters: 0.4000 0.6000
0.980
0.588
0.196
-0.196
-0.588
-0.980
-0.790 -0.444 -0.100 0.220 0.564 0.900
x
1
Domain 4
Parameters: 1.0000 1.0000
1.437
1.087
0.737
0.388
0.038
-0.311
-0.319 0.089 0.496 0.904 1.312 1.720
x
2
Domain 4
Parameters: 0.5000 1.0000
1.457
1.113
0.768
0.423
0.078
-0.266
-0.270 0.075 0.420 0.764 1.109 1.454
x


3
Domain 5
Parameters: 0.2000 0.3000
2.068
1.638
1.209
0.779
0.350
-0.080
-0.312 0.189 0.690 1.191 1.692 2.193
y
x
4
Domain 5
Parameters: 1.4000 0.5000
2.068
1.638
1.209
0.779
0.350
-0.080
-0.312 0.189 0.690 1.191 1.692 2.193
y
x
1
Domain 6
Parameters: 0. 0.
9.890
7.630
5.370
3.110
0.851
-1.409
-1.614 1.022 3.659 6.295 8.931 11.568
y
x
2
Domain 6
Parameters: -0.5000 1.0000
10.993
7.988
4.984
1.980
-1.025
-4.029
-2.528 1.678 5.885 10.091 14.297 18.503
y
x


3
Domain 7
Parameters: 0.4000 0.2000
0.658
0.491
0.325
0.158
-0.008
-0.174
-0.182 0.051 0.284 0.517 0.750 0.983
y
x
4
Domain 7
Parameters: 0.6000 0.1000
0.658
0.491
0.325
0.158
-0.008
-0.174
-0.182 0.051 0.284 0.517 0.750 0.983
y
x
1
Domain 8
Parameters: 1.0000 1.0000
2.456
1.874
1.291
0.709
0.127
-0.455
-0.456 0.126 0.709 1.291 1.873 2.455
y
x
2
Domain 8
Parameters: -0.1000 0.3000
0.337
0.162
-0.013
-0.187
-0.362
-0.536
-0.537 -0.362 -0.187 -0.013 0.162 0.336
y
x

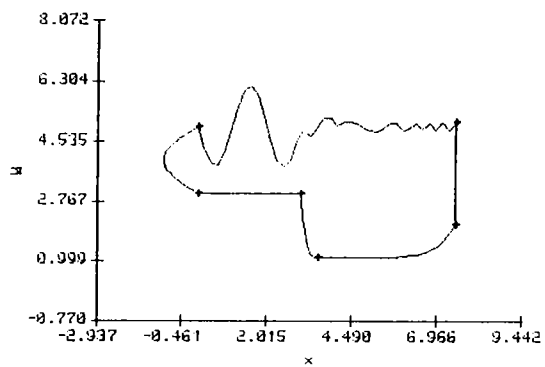
3

Domain 9
Parameters: 0.2000 0.2000



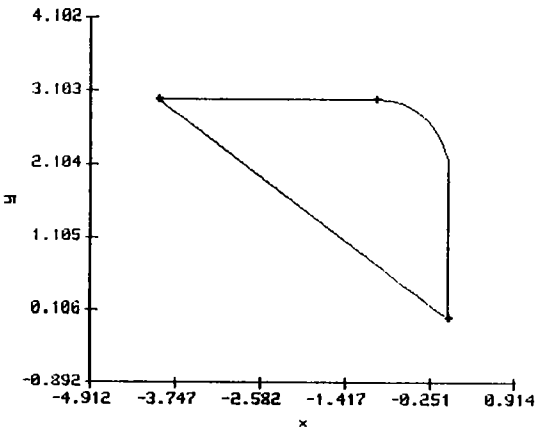
4

Domain 9
Parameters: 1.2000 -0.3000



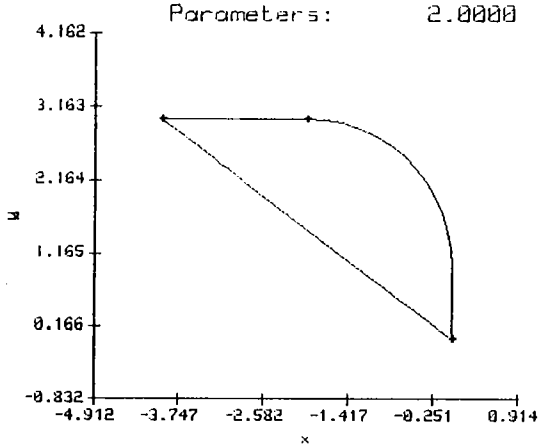
1

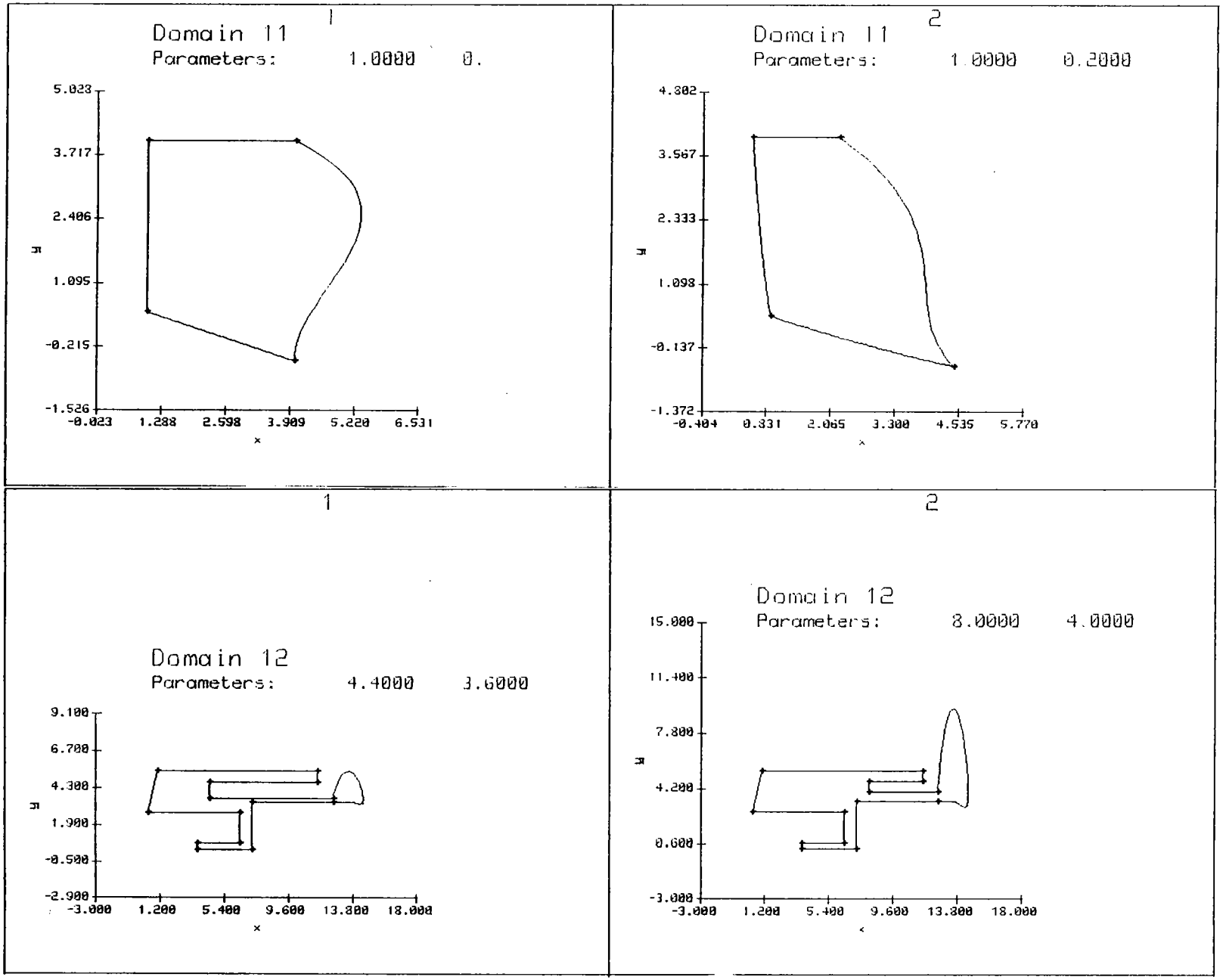
Domain 10
Parameters: 1.0000 0.

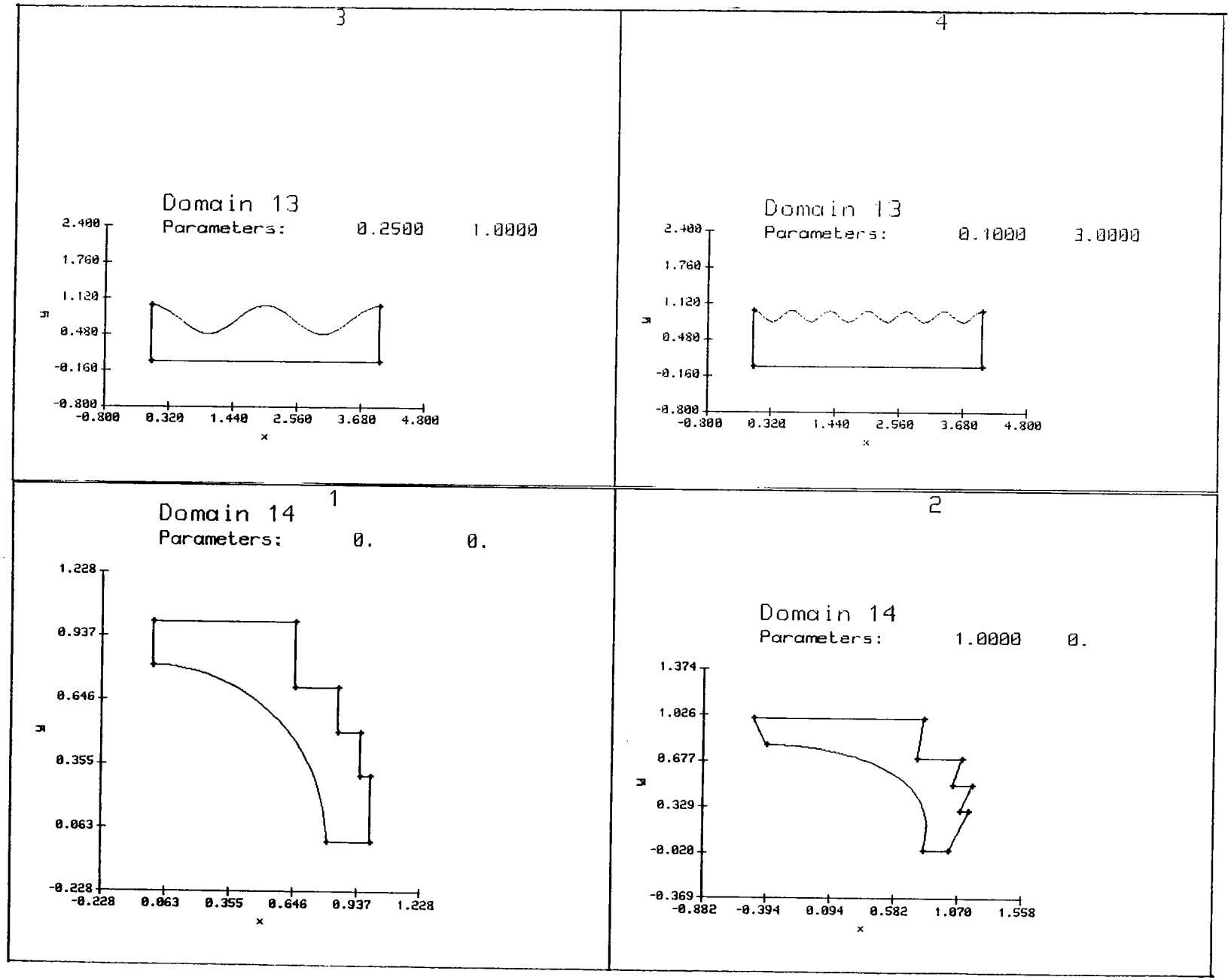


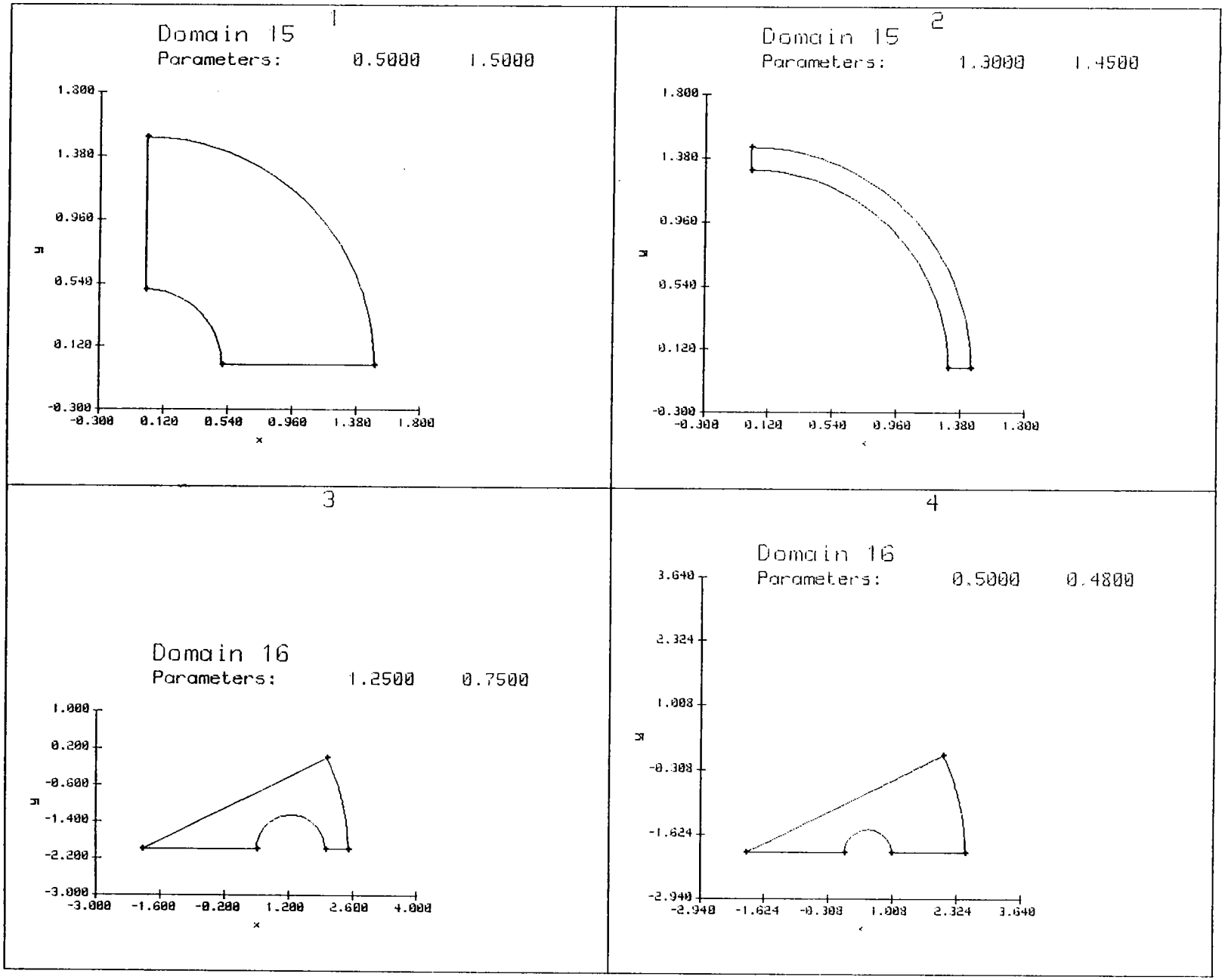
2

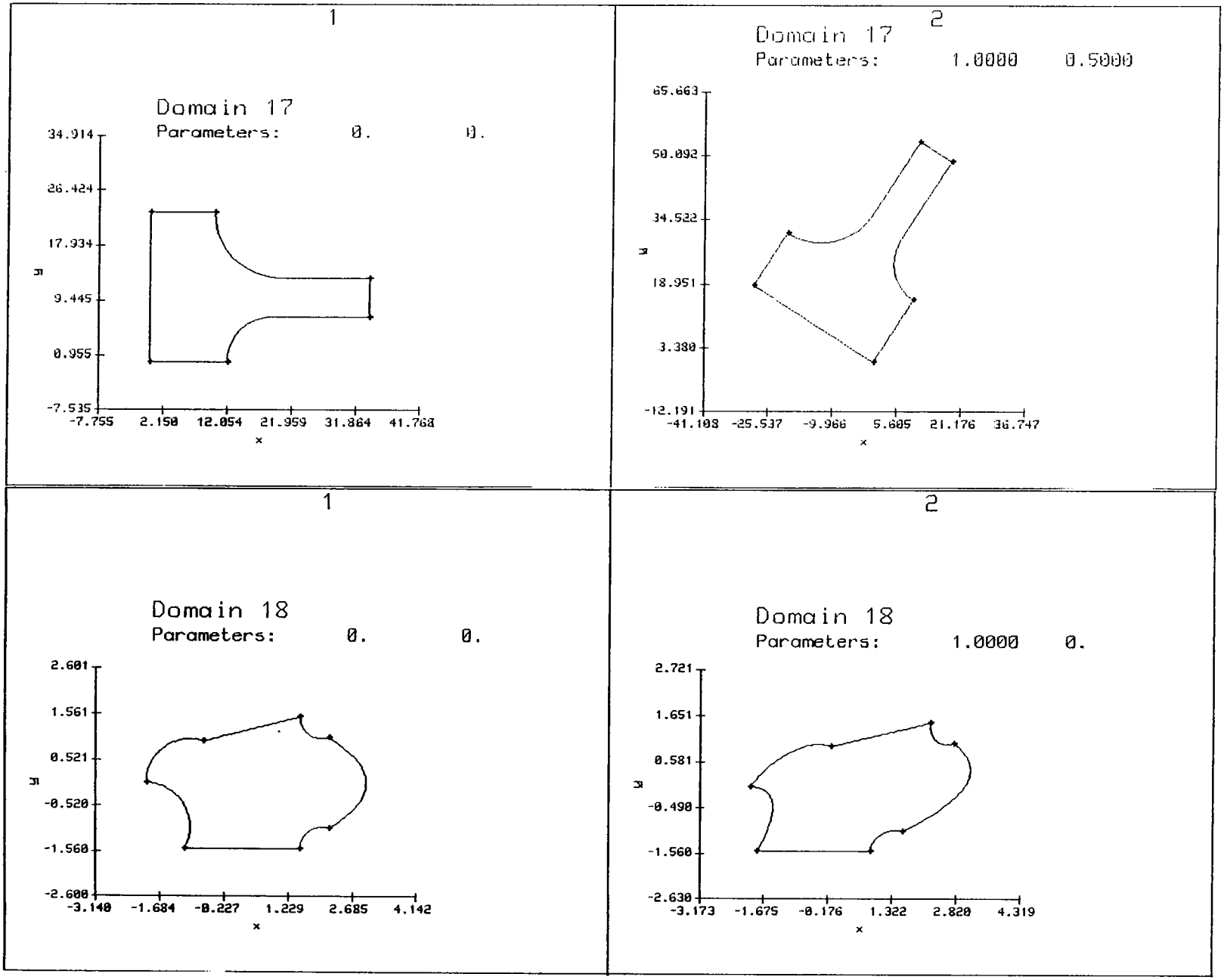
Domain 10
Parameters: 2.0000 0.

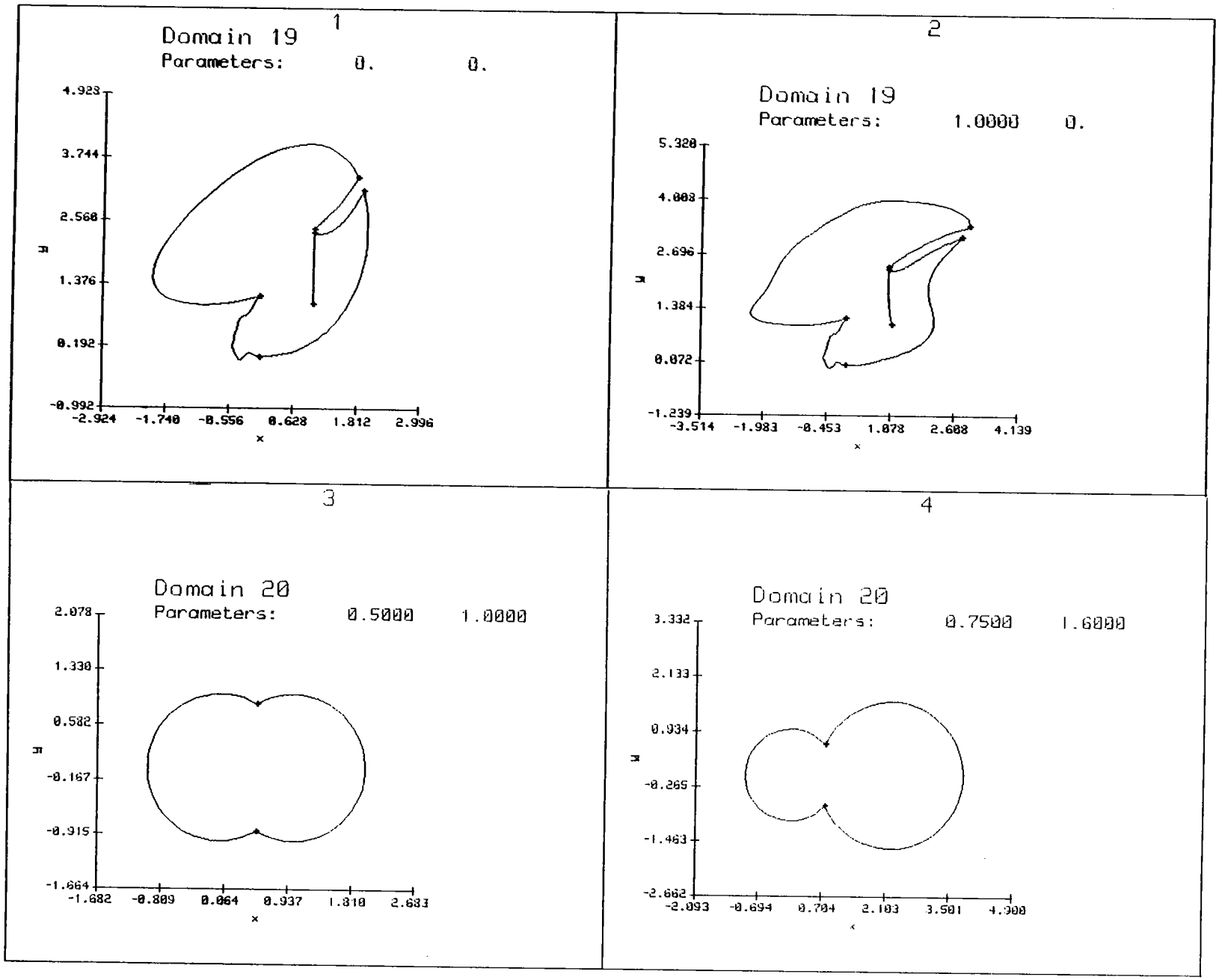


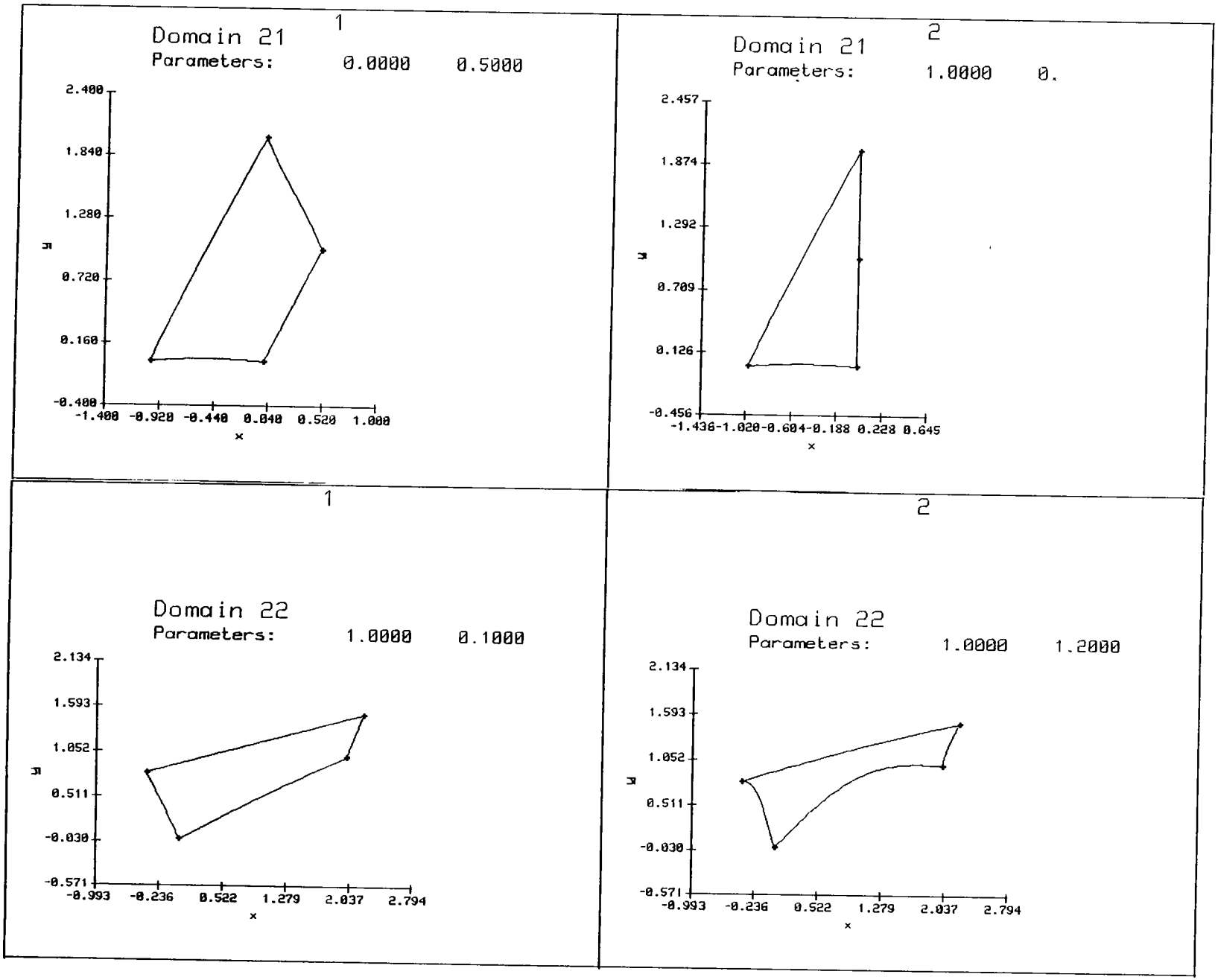

1
Domain 11
Parameters: 1.0000 0.
5.023
3.717
2.406
1.095
-0.215
-1.526
y
-0.023 1.288 2.598 3.909 5.220 6.531
x
2
Domain 11
Parameters: 1.0000 0.2000
4.302
3.567
2.333
1.098
-0.137
-1.372
y
-0.404 0.331 2.065 3.300 4.535 5.770
x
1
Domain 12
Parameters: 4.4000 3.6000
9.100
6.700
4.300
1.900
-0.500
-2.900
y
-3.000 1.200 5.400 9.600 13.800 18.000
x
2
Domain 12
Parameters: 8.0000 4.0000
15.000
11.400
7.800
4.200
0.600
-3.000
y
-3.000 1.200 5.400 9.600 13.800 18.000
x

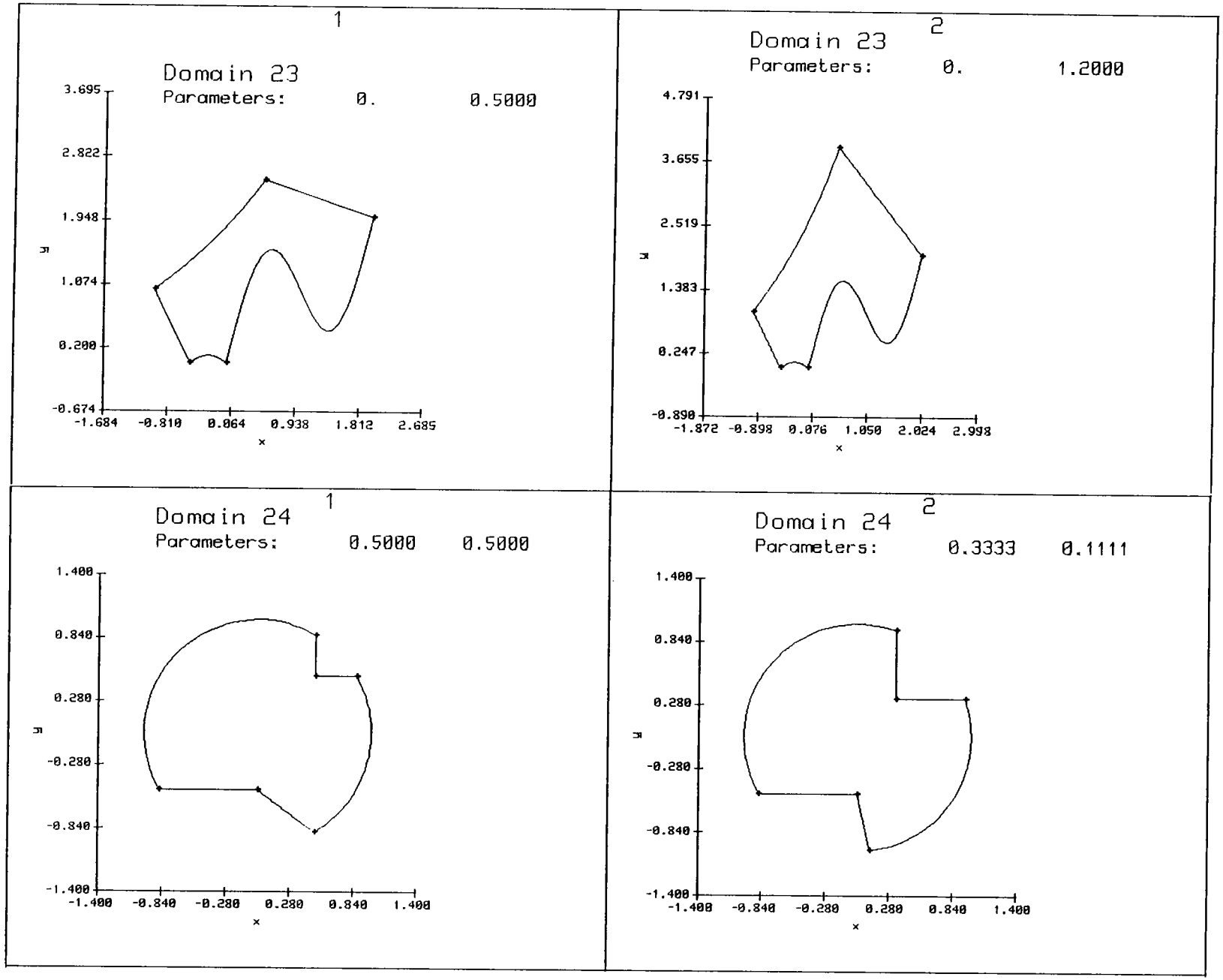

3
Domain 13
Parameters: 0.2500 1.0000
2.400
1.760
1.120
0.480
-0.160
-0.800
y
-0.800 0.320 1.440 2.560 3.680 4.800
x
4
Domain 13
Parameters: 0.1000 3.0000
2.400
1.760
1.120
0.480
-0.160
-0.800
y
-0.800 0.320 1.440 2.560 3.680 4.800
x
1
Domain 14
Parameters: 0. 0.
1.228
0.937
0.646
0.355
0.063
-0.228
y
-0.228 0.063 0.355 0.646 0.937 1.228
x
2
Domain 14
Parameters: 1.0000 0.
1.374
1.026
0.677
0.329
-0.020
-0.369
y
-0.882 -0.394 0.094 0.582 1.070 1.558
x

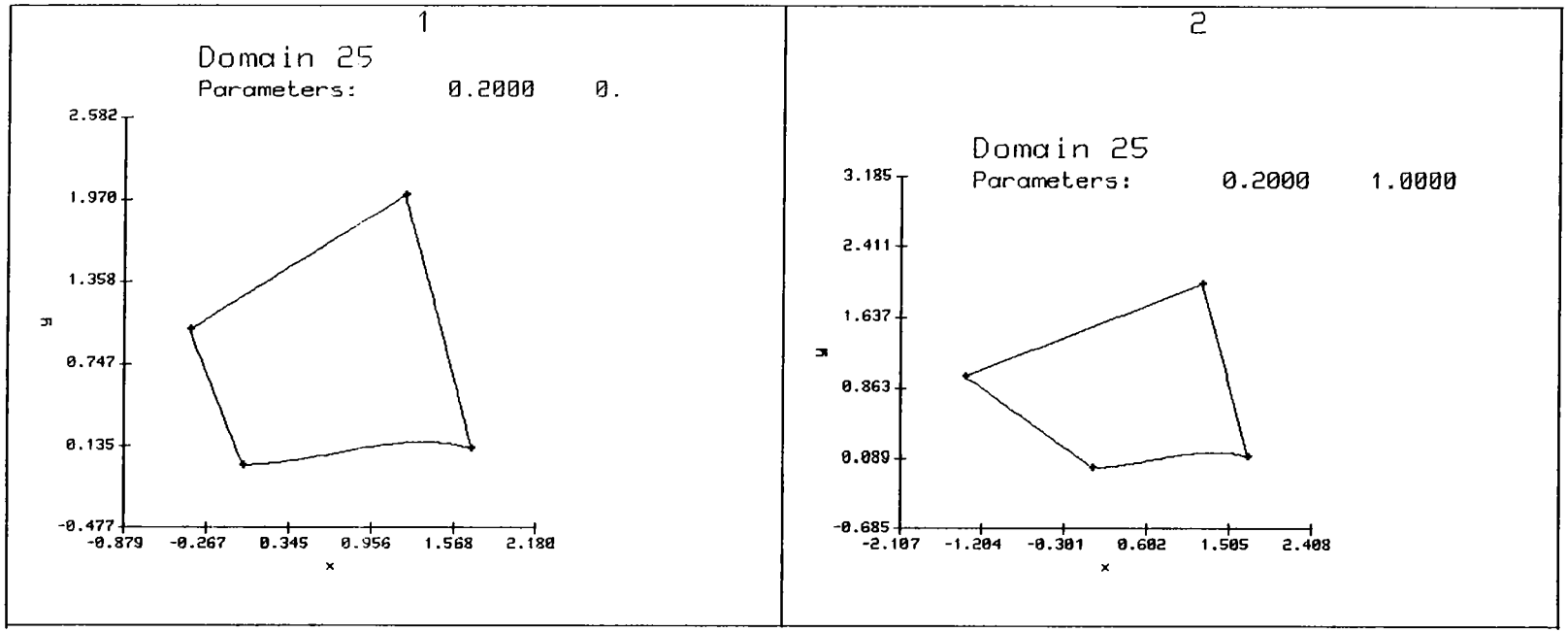

1
Domain 15
Parameters: 0.5000 1.5000
1.800
1.380
0.960
0.540
0.120
-0.300
y
-0.300 0.120 0.540 0.960 1.380 1.800
x
2
Domain 15
Parameters: 1.3000 1.4500
1.800
1.380
0.960
0.540
0.120
-0.300
y
-0.300 0.120 0.540 0.960 1.380 1.800
x
3
Domain 16
Parameters: 1.2500 0.7500
1.000
0.200
-0.600
-1.400
-2.200
-3.000
y
-3.000 -1.600 -0.200 1.200 2.600 4.000
x
4
Domain 16
Parameters: 0.5000 0.4800
3.640
2.324
1.008
-0.308
-1.624
-2.940
y
-2.940 -1.624 -0.308 1.008 2.324 3.640
x


1
Domain 17
Parameters: 0. 0.
34.914
26.424
17.934
9.445
0.955
-7.535
y
-7.755 2.150 12.054 21.959 31.864 41.768
x
2
Domain 17
Parameters: 1.0000 0.5000
65.663
50.092
34.522
18.951
3.380
-12.191
y
-41.108 -25.537 -9.966 5.605 21.176 36.747
x
1
Domain 18
Parameters: 0. 0.
2.601
1.561
0.521
-0.520
-1.560
-2.600
y
-3.140 -1.684 -0.227 1.229 2.685 4.142
x
2
Domain 18
Parameters: 1.0000 0.
2.721
1.651
0.581
-0.490
-1.560
-2.630
y
-3.173 -1.675 -0.176 1.322 2.820 4.319
x


1
Domain 19
Parameters: 0. 0.
4.928
3.744
2.560
1.376
0.192
-0.992
-2.924 -1.740 -0.556 0.628 1.812 2.996
x
y
2
Domain 19
Parameters: 1.0000 0.
5.320
4.008
2.696
1.384
0.072
-1.239
-3.514 -1.983 -0.453 1.078 2.608 4.139
x
y
3
Domain 20
Parameters: 0.5000 1.0000
2.078
1.330
0.582
-0.167
-0.915
-1.664
-1.682 -0.809 0.064 0.937 1.810 2.683
x
y
4
Domain 20
Parameters: 0.7500 1.6000
3.332
2.133
0.934
-0.265
-1.463
-2.662
-2.093 -0.694 0.704 2.103 3.501 4.900
x
y


1
Domain 21
Parameters: 0.0000 0.5000
2.400
1.840
1.280
0.720
0.160
-0.400
y
-1.400 -0.920 -0.440 0.040 0.520 1.000
x
2
Domain 21
Parameters: 1.0000 0.
2.457
1.874
1.292
0.709
0.126
-0.456
y
-1.436-1.020-0.604-0.188 0.228 0.645
x
1
Domain 22
Parameters: 1.0000 0.1000
2.134
1.593
1.052
0.511
-0.030
-0.571
y
-0.993 -0.236 0.522 1.279 2.037 2.794
x
2
Domain 22
Parameters: 1.0000 1.2000
2.134
1.593
1.052
0.511
-0.030
-0.571
y
-0.993 -0.236 0.522 1.279 2.037 2.794
x


1
Domain 23
Parameters: 0. 0.5000
3.695
2.822
1.948
1.074
0.200
-0.674
-1.684 -0.810 0.064 0.938 1.812 2.685
y
x
2
Domain 23
Parameters: 0. 1.2000
4.791
3.655
2.519
1.383
0.247
-0.890
-1.872 -0.898 0.076 1.050 2.024 2.998
y
x
1
Domain 24
Parameters: 0.5000 0.5000
1.400
0.840
0.280
-0.280
-0.840
-1.400
-1.400 -0.840 -0.280 0.280 0.840 1.400
y
x
2
Domain 24
Parameters: 0.3333 0.1111
1.400
0.840
0.280
-0.280
-0.840
-1.400
-1.400 -0.840 -0.280 0.280 0.840 1.400
y
x


1
Domain 25
Parameters: 0.2000 0.
2.582
1.970
1.358
0.747
0.135
-0.477
y
-0.879
-0.267
0.345
0.956
1.568
2.180
x
2
Domain 25
Parameters: 0.2000 1.0000
3.185
2.411
1.637
0.863
0.089
-0.685
y
-2.107
-1.204
-0.301
0.602
1.505
2.408
x